1983

The design and implementation of a tutorial program to perform symbolic mathematics

Richard Derek Otieno

College of Humanities and Sciences
Virginia Commonwealth University

This is to certify that the thesis prepared by Richard Derek Otieno entitled The Design and Implementation of a Tutorial Program to Perform Symbolic Mathematics has been approved by his committee as satisfactory completion of the thesis requirement for the Master of Science degree in Mathematical Sciences.

[redacted]

Dr. Richard E. Allan
Director of Thesis

[redacted]

Dr. James E. Ames, IV
Committee Member

[redacted]

Dr. C. Michael Lohr
Committee Member

[redacted]

Dr. Ena Gross
Committee Member

[redacted]

Dr. James A. Wood
Director of Graduate Studies

[redacted]

Dr. William E. Haver
Department Chairman

[redacted]

Dr. Elske V.P. Smith
Dean, College of Humanities
and Sciences

Dec. 12, 1983
Date

THE DESIGN AND IMPLEMENTATION OF A TUTORIAL PROGRAM TO PERFORM SYMBOLIC MATHEMATICS

A thesis submitted in partial fulfillment of
the requirements for the degree of Master of Science
in Mathematical Sciences at
Virginia Commonwealth University

by

Richard Derek Otieno

Richard E. Allan, Ph.D, Director
Associate Professor of Mathematical Sciences

Virginia Commonwealth University
Richmond, Virginia

December, 1983

ACKNOWLEDGEMENTS

The author wishes to give special thanks to Dr. Richard E. Allan for his guidance, advice, and patience and insight throughout the development of this paper. The author is also indebted to Dr. Lohr, Dr. Gross and Dr. Ames for their suggestions during the testing of the program on the types of responses that would be meaningful to the student, and also on the format that should be adapted.

The author also wishes to acknowledge the support, and the contributions of all family members and friends who assisted his education. Special thanks go to Erick and Elsa Onyango who influenced the author to join VCU's graduate program. Special thanks also go to my sister Monica Achieng Omaya. The financial support given by Frederic and Francoise Ateto is worthy of recognition in their support of the author during his education from 10th grade through first year graduate school. Without this support which was priceless and rare, none of this work would have been possible.

The author wishes to dedicate this work to his mother, Bernadette Okach Omaya for all the sacrifices that she has made for her children's education; and to his wife, Adhiambo Otieno for her patience, prayers and support during the preparation of this thesis.

Finally, special thanks are extended to Brenda Baskins for all the typing and assistance which made it possible for this paper to be completed in a timely manner.

TABLE OF CONTENTS

List of Figures

List of Tables

DESIGN AND IMPLEMENTATION OF A TUTORIAL PROGRAM TO PERFORM SYMBOLIC MATHEMATICS

ABSTRACT

The purpose of this thesis is to design and implement a program that could be used for a drill in symbolic mathematics. The scope of the program with respect to the range of problems that it solves is limited to selected types from elementary algebra, trigonometry, differential calculus, and integral calculus. The program is designed, not only to give solutions (answers), but also to provide several of the intermediate steps leading to the final result.

FORMAC73 and PL/I are used to implement the program. FORMAC73 is a system and a language for manipulating mathematical expressions, symbolically. The facilities of PL/I are available for program structure, loop control, testing, input and output operations, and manipulating the strings so as to get the intermediate steps.

The program is designed to provide supplementary assistance to give students drill, practice, and review several specific topics in the stated areas. The program may also be used by instructors to check or to provide answers to particular problems, sav for an examination or homework. The program does not attempt to diagnose the student's difficulties.

The main accomplishments of this paper will be to show the potential of using FORMAC73 with the existing facilities at Virginia Commonwealth University (VCU) to develop an extensive tutorial drill; particularly in arithmetic and algebra. The paper will also show that it is possible to develop a comprehensive CAI package in symbolic mathematics by using existing symbolic mathematics systems.

INTRODUCTION

Introduction:

Symbolic mathematics is one of the many tasks that the computer can perform. Although it is not as trivial as numerical mathematics, there are programs equally well-adaptable to the manipulation of algebraic expressions. In other words, the computer can work not only with numbers, but with more abstract symbols that represent numerical quantities.

In what follows, some of these programs are discussed, and some of their successful applications in research are cited. Next, the author briefly examines some of the Computer Assisted Instruction (CAI) work that has been done in some fields, notably chemistry, and numerical mathematics. The author then sets out to show that the power of the existing symbolic mathematics systems can be used in preparing CAI packages which can be used to assist in teaching courses in mathematics, notably the remedial courses. Depending on the sophistication of the package, even higher level mathematics students can get substantial learning aid beyond just getting the solution from the system. Such development efforts should be seen as complementing existing CAI packages based on numerical mathematical manipulations.

The usefulness of computer-algebra (performing symbolic mathematical manipulations on the computer) has been noted in several places in the literature. One that quickly comes to mind is the work of a French astronomer, Charles Delauney who set out to calculate the position of the moon as a function of time. This calculation demanded twenty years of his life to complete, spending ten years to do it the first time, and another ten years to check the correctness of the first calculations. His work was published in 1867. His calculations were not challenged until 1970 when Andre Deprit, Jacques Henrad, and Arnold Rom made the same calculations in twenty hours of computer time (14).

Computer Aided Instruction (CAI), or Computer Assisted Learning (CAL) are acronyms for well-written educationally effective tutorial programs for various subjects. The terms CAI, and CAL are used interchangeably in the literature to connote the same meaning, although the author will adhere to using CAI. There are many CAI packages that have been developed. Some have been developed by leading universities, while others have been prepared by a consortia of universities and sold to other educational institutions. Other small scale CAI programs have been developed by individual departments within various universities to respond to their immediate needs. For example, Professor Ronald D. Cain of the university of Kansas, at Lawrence prepared a CAI program for Aromatic Organic Synthesis in the Time Sharing Fortran (14). His objective was to promote a clearer understanding of organic syntheses by beginning organic chemistry students; and to do it within the department's budgetary constraints. He diagnosed that most students would gain a clearer understanding of the synthesis process if the instructor could show each student, personally, where he/she went wrong in the process. While this would be desirable, it is not always possible, particularly at a large university. One of the obvious approaches to such logistical problems in teaching is to develop a CAI program to handle the majority of the students' needs for confirming which synthesis steps, and in what sequence, are feasible for making a compound. Similarly, Professor Alfred J. Lata within the same university and department as Professor Cain developed an interactive time-sharing basic tutorial program sequence in electrochemistry (9). These two examples may be considered as small scale CAI development efforts in two high level languages - BASIC and FORTRAN to address the problems that were most immediate for this department. The author has reason to believe that there are similar situations at several other universities who cannot afford the most elaborate CAI languages such as PLANIT, COURSEWRITER, or the PLATO system. The author also believes that such problems and constraints are not unique to chemistry alone, but are also inherent in mathematics as well.

The National Science Foundation (NSF) has funded the development and distribution of many such packages through some well-established consortia. One such consortium is CONDUIT. CONDUIT is a consortium involving five universities, namely: University of Iowa, Dartmouth College, North Carolina University, The University of Texas, and University of Oregon. CONDUIT has developed CAI packages in almost all disciplines offered at most universities and colleges. The scope of this paper is limited to the development of CAI programs in mathematics, and so some of these programs will be briefly discussed.

The author had an opportunity in 1980 to review some of the mathematics packages developed by CONDUIT. All the ones in existence at that time which the author was aware of were in numerical mathematics. Working in the capacity of an academic programmer for Randolph-Macon College, the author implemented two mathematics CAI programs, written in CONDUIT BASIC on the Perkin-Elmer 3220 System. These two programs are called INTRFACE, and MATHPGM. INTRFACE consists of several modules. Each module performs an entirely different function. For example, one module's function is to generate different values of the sine function (some preset number of values), another module plots the values of a given function which the program generates based on some specified interval. This package has thirty-two modules all together. MATHPGM, on the other hand, is more sophisticated than INTRFACE. An example of MATHPGM, and INTRFACE are included in Appendix "A".

Once these two packages were implemented among several others in the various disciplines, the author held a workshop for respective faculty members to familiarize themselves with the packages, so that they could incorporate using the computer in their course syllabi.

The attitude of some of the members from the mathematics department was very discouraging. In particular, one instructor asked if any of these packages could help his students understand the limit concept in calculus, or provide them with sufficient practice with analytical differentiation, and integration. Some of the modules in

INTRFACE do numerical integration (trapezodial rule), and some numerical differentiation.

It was shortly after this workshop that the author conceived of a mathematics package that would do symbolic (or analytical) mathematics. The thesis director was very instrumental in helping shape up the objectives of this paper. After some investigation and consultation with him, it was decided that designing and implementing a program that does symbolic algebra would be sufficient proof that a more extensive tutorial system, or CAI program, could be developed. This program would not only do symbolic algebraic manipulations, but also solve equations symbolically on the computer, and possibly have other extensions. PL/I was almost the obvious language to use since the author knows it well, and, also because it has a preprocessor called FORMAC73, all of which are already available at VCU. FORMAC73 is a system and a language for manipulating mathematical expressions, symbolically.

The author has, therefore, set forth to investigate the feasibility of developing a symbolic mathematics tutorial program. Such a program will be developed in a limited scale as outlined in Chapter 2. Before this development, five other symbolic mathematics systems were investigated. These symbolic mathematics systems are: PICOMATH, muMATH, REDUCE, MACSYMA, and SCRATCHPAD. FORMAC73 is discussed in more details in Chapter I. It is not discussed in Appendix B as were the others because it is the one which the author used to implement a prototype tutorial drill in symbolic mathematics, and so it is included in the mainstream of the paper.

In general, the tutorial drill selects problem types and confines its generality to solving only problems which are subsets of the selected problem types. Specifically, the program presents a "menu" at the beginning to enable the user to select a topic. Each selection from the top level menu may also provide a more detailed menu of alternatives describing specific problem types that the user may wish to practice solving. For instance, under arithmetic/algebra, the user has the option to practice solving the following problem types: simplifying expressions - both binomials, and trinomials, adding and/or subtracting fractions, comparing two fractions for equality, to mention a few. The only exceptions arise when differentiation, and integration selections are made from the top level menu. These selections do not provide other menus. Instead, they are the routines which solve the problems, differentiation, and integration, respectively.

Since the system is modularly designed, it allows for extensions to include other problem types which are not currently provided for. Next, additions of new modules can be done with relative ease, requiring minimum changes in the existing nucleus (or main program). The change to the main program simply involves respective calls at the appropriate places, and a recompilation and a relinkage of the entire program. The conclusions and recommendations from this study are based on the system designed by the author, the author's experience with some of the existing CAI packages and literature on symbolic mathematics.

CHAPTER I

FORMAC73

FORMAC73 is a system and a language for manipulating mathematical expressions symbolically. As such, it takes character strings, interprets them as mathematical expressions, and carries out transformations and simplifications on them according to a user's program.

FORMAC73 is an improved, and an extended version of the Formula manipulation language and system FORMAC. FORMAC is an ancronym for Formula Manipulation Compiler. It may be regarded as an extension of PL/1 with two components added, namely:

(1) A preprocessor called "MINIMAC', that automatically translates FORMAC73 user programs into legal PL/1 program.

(2) A set of library routines that are automatically called thereafter to carry out mathematical transformations according to the user program and to produce results. The execution process of a FORMAC73 user program is illustrated in Figure 1.1.

Once it is signalled that the user program is a FORMAC73 program, it is fed to the preprocessor, MINIMAC. MIMIMAC declares all the necessary procedures, and functions as external, and generates call statements at the appropriate points in the user program to the relevant external procedures. An example of an original FORMAC73 source program is given in Figure 1.2. The sample program after the preprocessor is given in Figure 1.3.

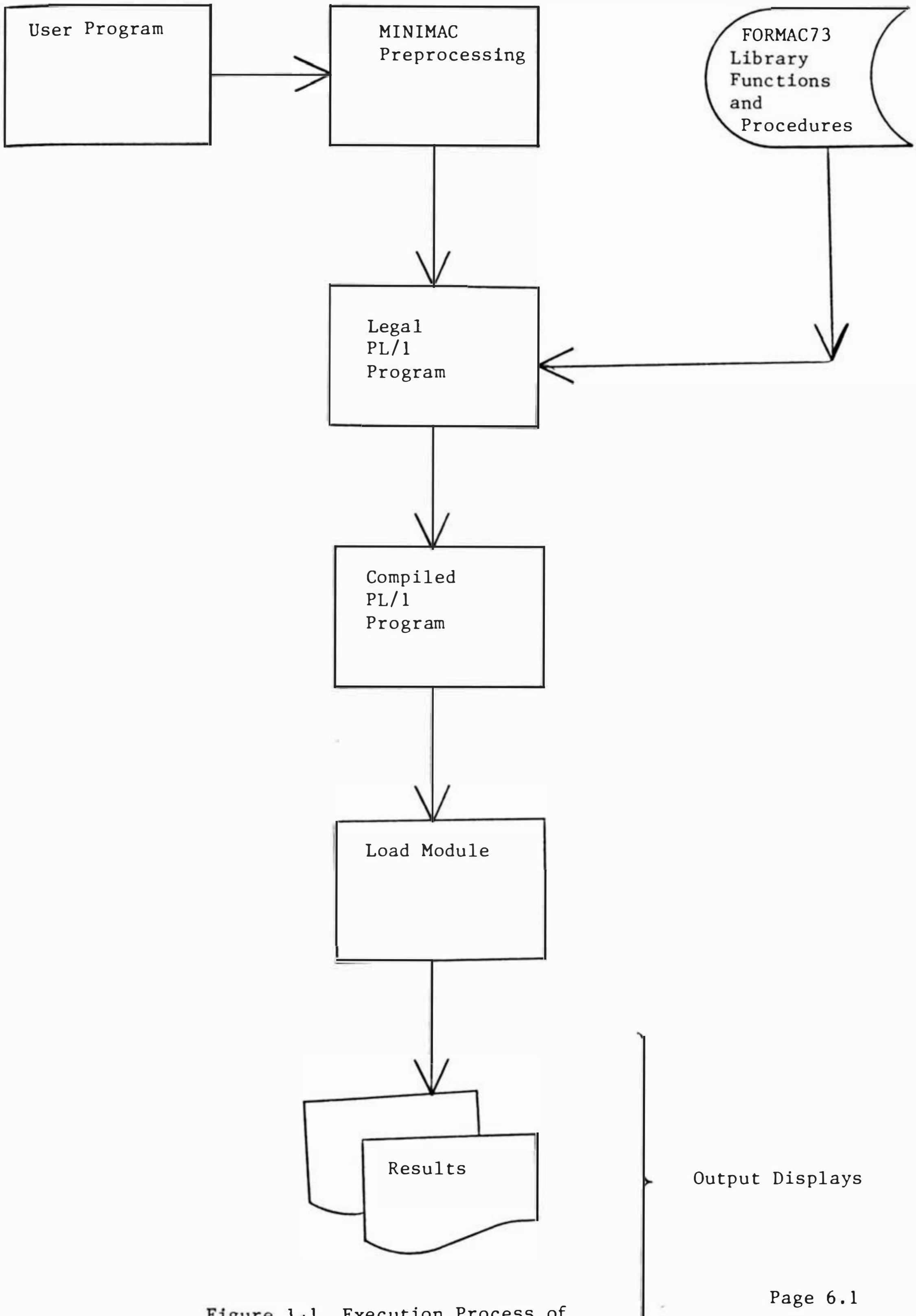


Figure 1.1 Execution Process of FORMAC73 User Program

INPUT TO KFA FORMAC PREPROCESSOR - VERSION 3 - GMI, DECEMBER 1976

```
LEGEND:  PROC OPTIONS (MAIN);
         FORMAC_OPTIONS;

            OPTSET(EXPND;LINELENGTH=72);

            PUT PAGE;

/*   COMPUTE LEGENDRE POLYNOMIALS BY METHOD # 1                          */

            DO N = 0 TO 10;
               LET (TERM2 = 2**"N"*FAC("N"));
               LET (P("N") = DERIV((X**2-1)**"N",X,"N")/TERM2);
               PRINT_OUT(P("N"));
            END;

/*   COMPUTE LEGENDRE POLYNOMIALS BY METHOD #2                           */

            LET (Q(0) = 1);
            LET (Q(1) = X);

            DO N = 2 TO 10;
               LET (N = "N");
               LET (Q(N) = (2*N-1)/N*Q(N-1)-(N-1)/N*Q(N-2));
               PRINT_OUT(Q("N"));
            END;

/*        THAT P(N) = Q(N) AND PRINT OUT THE RESULTS                     */

            PUT SKIP EDIT ('LEGENDRE POLYNOMIALS') (X(10),A);
            PUT SKIP (3);

            DO N = 0 TO 10;
               LET (N = "N");
               IF ( IDENT(P(N);Q(N)) )THEN
                  PRINT_OUT(P(N));
               ELSE
                  STOP;
            END;

        END LEGEND;
```

Figure 1.2

```
                 SOURCE LISTING


   STMT LEV NT


      1       0  LEGEND:  PROC OPTIONS (MAIN);                                                         1
                                                                                                       2
                /* DECLARES FOR RUN-TIME ROUTINES.         */                                          2
      2    1  0 DECLARE                                                                                2
                 DENPMC1 ENTRY(CHAR(*)),                                                               2
                 DENPMC2 ENTRY(CHAR(*)),                                                               2
                 DENPMC3 ENTRY(FIXED BIN(31),FIXED BIN(31)),                                           2
                 DENPMC4 ENTRY(CHAR(*)),                                                               2
                 DENPMC7 ENTRY(CHAR(*)),                                                               2
                 DENPMC8 ENTRY(CHAR(*)),                                                               2
                 DENPMC9 ENTRY(CHAR(*)) RETURNS(BIN FLOAT(53)),                                        2
                 DENPMCA ENTRY(CHAR(*)) RETURNS(BIN FIXED(31)),                                        2
                 DENPMCB ENTRY(CHAR(*)) RETURNS(BIN FIXED(31)),                                        2
                 DENPMCC ENTRY(CHAR(*)) RETURNS(BIN FIXED(31)),                                        2
                 DENPMCD ENTRY(CHAR(*),BIN FLOAT(53)),                                                 2
                 DENPMCE ENTRY(CHAR(*),BIN FIXED(31)),                                                 2
                 DENPMCF ENTRY(CHAR(*),CHAR(*)) RETURNS(BIT(1)),                                       2
                 DENPMCG ENTRY(CHAR(*),CHAR(*)) RETURNS(BIT(1)),                                       2
                 DENPMCH ENTRY,                                                                        2
                 DENPMCL ENTRY(CHAR(*)) RETURNS(POINTER),                                              2
                 DENPMCM ENTRY(CHAR(*),ENTRY);                                                         2
                                                                                                       2
      3    1  0 CALL DENPMC3(03,0);                                                                    4
      4    1  0 CALL DENPMC3(00,72);                                                                   4
      5    1  0             PUT PAGE;                                                                  6
                /*   COMPUTE LEGENDRE POLYNOMIALS BY METHOD # 1                         */             8
      6    1  0             DO N = 0 TO 10;                                                           10
      7    1  1 CALL DENPMC1('TERM2=2**'||N||'*FAC('||N||')');                                        11
      8    1  1 CALL DENPMC1('P('||N||')=DERIV((X**2-1)**'||N||',X,'||N||')/TERM2');                  12
      9    1  1 CALL DENPMC2('P('||N||')');                                                           13
     10    1  1             END;                                                                      14
                /*   COMPUTE LEGENDRE POLYNOMIALS BY METHOD #2                          */            16
     11    1  0 CALL DENPMC1('Q(0)=1');                                                               18
     12    1  0 CALL DENPMC1('Q(1)=X');                                                               19
     13    1  0             DO N = 2 TO 10;                                                           21
     14    1  1 CALL DENPMC1('N='||N||'');                                                            22
     15    1  1 CALL DENPMC1('Q(N)=(2*N-1)/N*Q(N-1)-(N-1)/N*Q(N-2)');                                 23
     16    1  1 CALL DENPMC2('Q('||N||')');                                                           24
     17    1  1             END;                                                                      25
                /*          THAT P(N) = Q(N) AND PRINT OUT THE RESULTS                  */            27
     18    1  0             PUT SKIP EDIT ('LEGENDRE POLYNOMIALS') (X(10),A);                         29
     19    1  0             PUT SKIP (3);                                                             30
     20    1  0             DO N = 0 TO 10;                                                           32
     21    1  1 CALL DENPMC1('N='||N||'');                                                            33
     22    1  1                IF ( DENPMCG('Z9999997=P(N)','Z9999998=Q(N)') )                        34
                 THEN                                                                                 35
```

Page 6.3

Figure 1.3

$P(0) = 1$

$P(1) = X$

$P(2) = 3/2\ X^{2} - 1/2$

$P(3) = 5/2\ X^{3} - 3/2\ X$

$P(4) = 35/8\ X^{4} - 15/4\ X^{2} + 3/8$

$P(5) = 63/8\ X^{5} - 35/4\ X^{3} + 15/8\ X$

$P(6) = 231/16\ X^{6} - 315/16\ X^{4} + 105/16\ X^{2} - 5/16$

$P(7) = 429/16\ X^{7} - 693/16\ X^{5} + 315/16\ X^{3} - 35/16\ X$

$P(8) = 6435/128\ X^{8} - 3003/32\ X^{6} + 3465/64\ X^{4} - 315/32\ X^{2} + 35/128$

$P(9) = 12155/128\ X^{9} - 6435/32\ X^{7} + 9009/64\ X^{5} - 1155/32\ X^{3} + 315/128$

X

$P(10) = 46189/256\ X^{10} - 109395/256\ X^{8} + 45045/128\ X^{6} - 15015/128\ X^{4}$

$+ 3465/256\ X^{2} - 63/256$

$Q(2) = 3/2\ X - 1/2$

$Q(3) = 11/6\ X - 5/6$

$Q(4) = 25/12\ X - 13/12$

$Q(5) = 137/60\ X - 77/60$

$Q(6) = 49/20\ X - 29/20$

$Q(7) = 363/140\ X - 223/140$

$Q(8) = 761/280\ X - 481/280$

$Q(9) = 7129/2520\ X - 4609/2520$

$Q(10) = 7381/2520\ X - 4861/2520$

LEGENDRE POLYNOMIALS

```
STMT LEV NT

             CALL DENPMC2('F(N)');                                       35
  23   1  1   ELSE                 STOP;                                 37
  24   1  1                  END;                                        38
  25   1  0              END LEGEND;                                     40
```

Facilities Provided by FORMAC73

All the symbolic manipulations that FORMAC73 does on mathematical expressions are made possible through the following features:

1. FORMAC Primitives
2. FORMAC Commands
3. FORMAC Functions
4. FORMAC - PL/1 Interface.

Each of these features is briefly discussed below.

FORMAC Primitives

All existing FORMAC Primitives may be grouped in the following five categories:

(a) Constants

(b) Variables

(c) Operators

(d) Expressions, and

(e) Chains

Constants

There are four types of Constants in FORMAC73, namely: integers, rational numbers, floating-point numbers, and system constants. Integers range in magnitude from a 1-digit integer to 2295-digit integer (5). Internally, integer numbers having a magnitude of 31623 are stored as a linked list of words where each word contains 9-decimal digits. It should be pointed out that storage constraints, particularly the floating point constraints are machine dependent. The limits quoted here apply to the IBM 360/370 series.

Rational numbers are externally represented as a division of two integers, i.e. a/b. Arithmetic operations on them are carried out using "rational mode arithmetic". For example,

$$1/2 + 1/4 = 3/4$$

$$8 * 6/12 = 4$$

$$8 * 7/12 = 14/3$$

Floating-point numbers range in magnitude from 5.4×10^{-79} to 7.2×10^{75}. Floating-point constants are maintained in double precision. They may combine with integers and rational numbers to always give a floating point number. For example,

1.0 * 1/3 = 0.3333333,

but 1 * 1/3 = 1/3

Clearly the results depend on the mode of the operands. There are three reserved names used as system constants. These are as follows:

#P represents 3.14159265

#E represents 2.71828183

#I represents SQRT (-1)

FORMAC73 Variables

FORMAC73 Variables are of primary interest to the batch user who has to prepare a user program, particularly those users who develop more sophisticated batch programs. FORMAC Variables have universal scope by default. Prate discusses the variuos scope rules governing variables more extensively(15). They can, however, be localized in scope. There is a distinction between FORMAC Variables and PL/1 Variables, even those variables that have identical names.

There are four types of variables, namely: atomic variables, assigned variables, function variables, and $-variables. An atomic variable is one which has not been given a value. Variables that have not been assigned values may be initialized by atomizing them. An assigned variable is one that has appeared on the left hand side of the equal sign, and with corresponding value on the right hand side of the equal sign. A function variable identifies an unspecified function. hey are mostly used in pattern matching operations along with FORMAC's other commands, or functions, i.e. EVAL and REPLACE. A $-variable is one which is free with respect to binding. It is not bound to any specific value. It is associated with a value only temporarily, and only during the evaluation process. A more detailed account and definition of variable binding is given by Pratt (15).

Pseudo-variables are only certain predefined functions that are allowed only on the left hand side of the equal sign of an assignment statement. They take a single argument, and serve to define a property of that argument. There are only four pseudo-variables supported by FORMAC. These are as follows:

FNC - to make FNAME a formula function (FNAME is a FORMAC NAME)

DIFF - to define derivatives for a function, "VAR FNAME"

STACK - to make FNAME a stack and to stack values

QUEUE - to make FNAME CHAIN and to queue values

FORMAC uses the usual mathematical operators, namely: **, *, /, +, and - for exponentiation, multiplication, division, addition, and subtraction, respectively. he expressions for FORMAC are syntactically defined using the Backus Naur Form (BNF), as follows (5; pp.13):

<FEXPR>::=<TERM>+<FEXPR>|<FEXPR>-<TERM>|<TERM>

<TERM>::=<TERM>*<FACT>|<TERM>/<FACT>|
<FACT>

<FACT>::=<BASE>**<FACT>|<BASE>

<BASE>::=<PRIM>|:<RPIM>|<PRIM>

<PRIM>::- <FEXPR>|<FORMAC CONSTANT>|
<FORMAC VARIABLE>|<FORMAC FUNCTION>

The entire FORMAC expression gets simplified at evaluation, except for when $-variable are used to signal delayed evaluation.

The Chain function is especially useful in building argument lists for FORMAC functions and routines. The queue is useful in building up an n-element chain dynamically.

FORMAC Commands

FORMAC features and capabilities are utilized via commands. Altogether, there are eleven types of commands which allow the user to take advantage of FORMAC's capabilities. Probably the most basic of all these commands are FORMAC-OPTIONS, Assignment Statement, and PRINT-OUT Statements. FORMAC-OPTIONS must precede the occurrence of any other FORMAC statements or features, and it may appear only once. It serves to signal that the program being executed is a FORMAC program; and so the preprocessor MIMIMAC must be invoked. It is a non-executable command.

The FORMAC assignment statement LET and its variations facilitate the simplification of expressions on the right hand of the equal sign, and their assignment to the name on the left hand side. Other variations refer to the pseudo variables FNC, DIFF, STACK, and QUEUE, which have been previously discussed. Other variations allow for the handling of PL/I strings. The PRINT-OUT command is executed in two steps: First as a LET statement, and then, second, the name and the value are printed out onto SYSPRINT. The output is edited for easy reading, depending on whether or not the edit option is activated.

FORMAC Option settings are established to give the programmer some control over the manner of internal evaluation, simplification and external representation of FORMAC expressions. Once set, an option has universal scope for all procedures of a program and remains in effect until changed by a later option setting. There are two types of options: executable options and non-executable options. Executable options are controlled by on - off settings during the program execution. Non-executable options are set at compile time, and so they cannot be altered during program execution.

Freeing of space may be considered as serving two purposes. It frees up some space as well as initializes FORMAC variable of a particular type. Next, FORMAC provides a feature that will free space by making a provision to save the value of the left hand side variables into secondary storage. This feature is very time consuming, and it should be used discriminately. FORMAC allows the user to convert from a PL/1 numeric variable or expression into a FORMAC variable. It also permits the user to convert a PL/1 variable or expression to a FORMAC integer and assign as it as the value of a FORMAC Variable.

FORMAC also allows the programmer to convert a FORMAC expression into an unedited PL/1 String. This is often necessary for the breakdown of expressions and for getting the intermediate steps.

CONVERT which must appear after FORMAC-OPTIONS, but before any FORMAC Macros is used to allow the programmer to have FORMAC send numeric values to PL/1 variables, or to receive numeric values from PL/1. The command CONVERT must appear after FORMAC options but before any FORMAC macros are used. This feature is set up to take effect on those FORMAC variables enclosed in double quotes.

FORMAC affords the user the flexibility to write his own functions and procedures. These functions and procedures can accept actual parameters, or they may be parameterless. They may also use the LOCAL command to denote those variables which are to be considered local to the function or procedure. Localize command generates variables dynamically. In other words, whenever the command is executed, it will erase the current values, if any, that the variables had prior to execution of the corresponding previous localize command.

FORMAC also makes the provision for punching the values of FORMAC Variables on to cards or into a punch-file. This portion of FORMAC is not always loaded. Its use results in an overhead of 25K bytes.

FORMAC provides several other functions. Three types of functions may be distinguished, namely: function operators, formula functions, and function procedures. For function operators, the name of the function is treated and kept internally as an operator (e.g. SIN, just like +, - , or **). There are built-in rules for simplifying, differentiating, and evaluating such functions. The user cannot introduce new function operators. The value of formula functions is defined by a single closed-form FORMAC expression, which is stated in terms of formal parameters. Evaluation is by substituting the actual argument values for the formal parameters. The defining expression may reference any other function; recursive definitions are disallowed in general. Function procedures are defined by a procedure and are the most general type of function. The procedure returns a FORMAC expression as its value; it may have side-effects (101). The procedure may use local variables and may have recursive definitions.

Finally, FORMAC provides enough functions to support the PL/1 - FORMAC interface. It is through these facilities that the user's program is able to provide some intermediate steps and to supply input character strings into FORMAC functions for evaluation.

Limitations of FORMAC73

FORMAC73 is not an intelligent system. It is a symbolic processor that provides several but fixed number of functions and commands to manipulate mathematical expressions. These functions are called into the user program by the preprocessor. Thereafter, the program is executed as a PL/1 program. Through some sophisticated programming, the user can force out some intermediate steps from the mathematical manipulations which FORMAC provides the user. FORMAC73 is even more limited in interactive mode as to the amount of intermediate results that one can expect. For example, to evaluate $\frac{x^3 + x^2}{(x + 1)}$, FORMAC must be told what to do , and in so doing, some generality is lost. Next, to compute the second derivate of $y = x^2$, the first derivation must be forced out by the user program, otherwise, FORMAC will only display the final result. FORMAC73 in general was designed for taking in a mathematical expression, evaluating it using fixed rules, and then displaying the results. This makes it difficult to use FORMAC as a learning tool, unless a program is designed to take advantage of FORMAC73 capabilities and provide learning features as well.

CHAPTER II

PROGRAM DEVELOPMENT AND TECHNIQUES

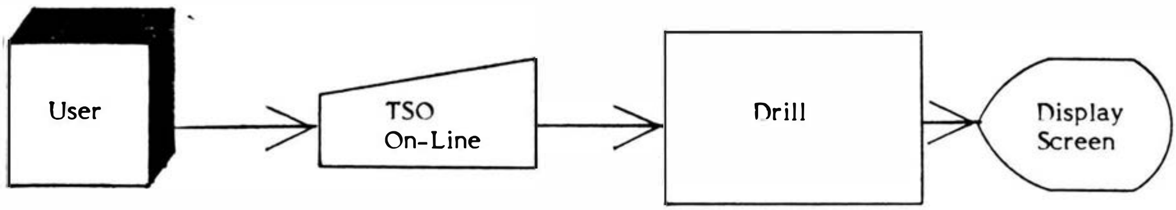
A. Introduction and Overview

In developing this program, the author has adhered to the techniques that assist in developing programs which are easy to read and modify. The program was modularly designed and coded using structured programming concepts. Since structured programming can take on a variety of meanings, the author has defined the one which was adopted in this development.

Under the "program (module) coding" section, syntactic conventions for each module format is identified, structured programming is defined, program style and variable/data usage are established.

How the system operates is shown in the two types of processing. Figures 2.1.1 and 2.1.2 are used to illustrate how the system operates for both interactive and batch processing, respectively. The only difference between the two figures is the I/O devices. The processing logic is the same. It requires the user to enter a problem to be solved and for the system to accept the problem, process it, and then generate the correct solution, along with the intermediate steps whenever possible.

TSO INTERACTIVE EXECUTION



Solutions to individual problems with intermediate results. This will also display the approprite prompts.

FIGURE 2.1.1

Here, the system will allow the user to solve problems interactively until the user decides to terminate the session.

BATCH EXECUTION VIA CARD READER

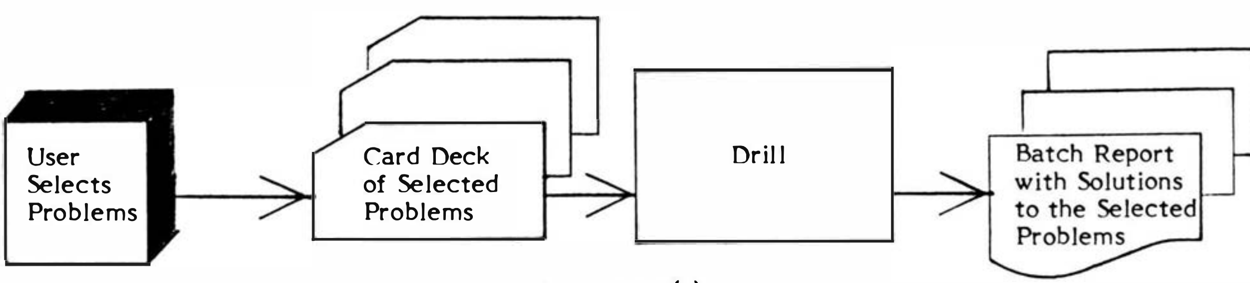


Figure 2.1.2(a)

BATCH EXECUTION VIA A DATA FILE

Here, the data set will have the same responses that would otherwise be provided to the system interactively. The reports will be exactly the same as those that would be produced via Figure 2.2.(a).

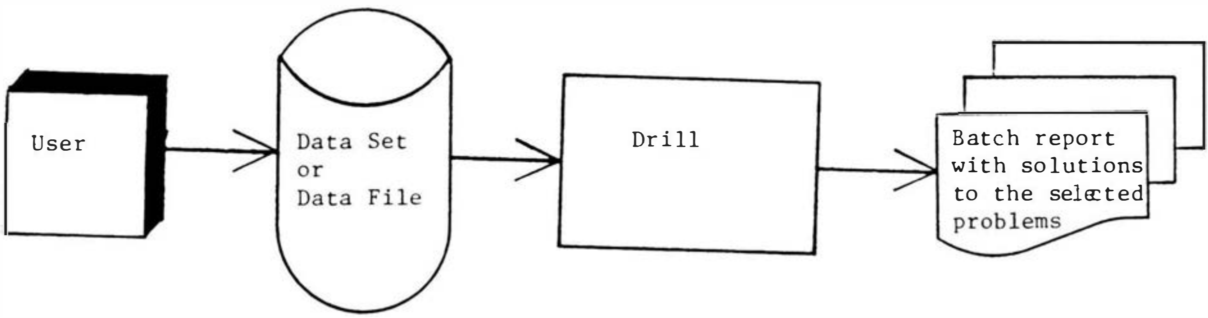


Figure 2.1.2 (b)

Requirement Specifications and Analysis

The program which the author proposes to develop must meet several criteria. It must be able to:

1. accept a problem of a given type in the appropriate format. The types of problems that will be solved have been selected from the following areas:

   (a) elementary algebra,

   (b) intermediate algebra,

   (c) trigonometry,

   (d) differential calculus, and

   (e) integral calculus.

2. perform the necessary transformations and simplifications.

3. provide a solution to the given problem along with the intermediate steps.

4. use standard I/O devices which are already available to the Department of Mathematical Sciences at VCU.

5. allow for ease of extension to include other problem selections. Since this program is developed to solve only certain selected problems from the five areas listed, this last requirement is a crucial one, especially if a more elaborate program is to be developed based on this program.

2. Feasibility Study

This section will analyze the plausibility of the requirements laid out in the previous section. For purposes of clarity, the analysis will be done step-wise (portion by portion).

1. accept a problem of a given type in the appropriate format.

Since the solution of the problems will be done symbolically, all the problems can be read in as character strings. They can then be broken down, as necessary, using PL/1 character string functions. Since there are several problem types, all these types must be identified. Once that is done, sufficient processing can be done on the incoming string to enhance further manipulations.

2. perform the necessary transformations and simplications.

This requirement is the heart of the entire program and also the most difficult. Performing transformations on mathematical expressions on the computer requires extensive character string pre-processing, before effecting mathematical transformations and simplications. The original problem must be decomposed, parsed and manipulated by established mathematical procedures. For example, as Hanson and Russo point out, to perform rational arithmetic, it is necessary to convert the rational number into a floating point number, perform floating point arithmetic, and then convert the resulting floating point number into the equivalent rational number (16).

Investigation has revealed that PL/1 has a preprocessor which assists in manipulating mathematical expressions. This preprocessor is called FORMAC73. Even though FORMAC73 provides limited control structures, further study shows that there exists sufficient interface with PL/1 to allow one to use PL/1 control structures. It is therefore possible to read a mathematical expression as a PL/1 character string, scan it, brake it down into designated primitive components, and then use FORMAC73 facilities to perform the necessary mathematical transformations and simplications.

3. provide a solution to the given problem along with the intermediate steps.

Since most of the transformations and the simplications can only be effected by the facilities provided by the preprocessor, the number of intermediate steps that the proposed program can produce is limited by:

(a) how the preprocessor performs its functions for the different problems, and

(b) the interface functions between PL/1 and FORMAC73.

Two examples will be used to further clarify. First, in differentation, there is no provision for getting the intermediate steps. A mathematical function translated into a FORMAC73 expression is differentiated when the appropriate (differentiating) function is invoked and the correct answer is returned. Therefore, it is not possible to get hold of the intermediate steps. Second, in adding two fractions, one may easily identify the numerator, the denominator, and the lowest common denominator. In this case, the required intermediate steps may be easily obtained and displayed. The number of intermediate steps that this program proposes to produce will be limited by the preprocessor. No attempts will be made to modify or to enhance the preprocessor in order to get all the desired intermediate steps.

4. use standard I/O devices which are already available to the Department of Mathematical Sciences at Virginia Commonwealth University.

Preliminary investigation has demonstrated that standard I/O devices are adequate for performing the tasks that this program has set forth to do. Standard I/O devices are used by the author to mean a regular printer, and a standard card reader(any card reader that would read an 80-column card). Alternately, facilities that can maintain and/or transmit files (or data sets) in card image format may also be used. In fact, the entire program will be written on the HP 3000 which is such a facility. The program along with instream data is transmitted to the host computer, in this case AMDAHL 470, as a batch job.

It is, however, more desirable to run the program interactively. The author will prepare it for interactive execution on TSO (Time Sharing Option), and also for batch execution. The devices and facilities that support interactive execution are not considered standard I/O devices by the author's definition.

Use of this program will require use of some symbolism that may not be readily familiar to a computer neophyte which may be inconveniencing at first, but this should soon be overcome. The new symbolism include using "*" to denote multiplication, "**" to denote raising to a power in order to unambiguously specify expressions using a one-dimensional format compartible with the limitations of standard keyboards and displays. Division, addition, and subtraction all require the use of their usual symbolism of "/", "+", and "-", respectively. Table 2.1 shows expressions as they are written mathematically, and how they must be entered into the computer.

Table 2.1

MATHEMATICAL EXPRESSION	FORMAT FOR ENTRY INTO THE COMPUTER
$3x + 2$	3 * x + 2
$3x^2 + bx + c$	3 * x ** 2 + b * x + c
$3x^2 + c$	(3 * x ** 2 + c) ** (1/2)
$3x^3$	3 * x ** 3
$(y2)^{1/3}$ or $\sqrt[3]{y^2}$	(y ** 2) ** (1/3)

What is shown as entered into the computer in Table 2.1 may be entered using any standard input device.

The output to the printer would appear in the same format that the expression enters the computer. Further processing can be done to enhance readability. FORMAC73 does some preprocessing of the output and Table 2.2 shows how the various mathematical expressions may be expressed.

Table 2.2

MATHEMATICAL EXPRESSION	SAME EXPRESSION ON A STANDARD PRINTER
$3x + 2$	FEXPR = 3x + 2
$3x^2 + bx + c$	FEXPR = $3x^2 + bx + c$
$\sqrt{(3x^2 + c)}$	FEXPR = $(3x^2 + c)^{1/2}$
$\frac{b - \sqrt{b^2 + 4ac}}{2a}$	FEXPR = $1/2(b - (4ac + b^2)^{1/2}a$

"FEXPR" is used to indicate that there is always a name on the left hand side. The name is limited to seven characters, and it must start with an alphabetic character. The user need not concern himself/herself with this detail because the program takes care of it automatically. It is mentioned to show that there is a limitation as to how descriptive the author could be in the selection of the name on the left hand side, and still get an edited output, such as shown above.

5. allow for ease of extension to include other problem selection.

This can be achieved through modular design. The program is designed in this manner so that the newly added modules will not interfere with the ones that have been implemented. Furthermore, no major changes would be required on this program other than the addition of the calls to the newly created modules.

B. Program Design

This section will show the detailed design of the proposed system. The following areas will be discussed in more details:

1. Modules
2. Top down design
3. Module Specification
4. Module implementation

1. Modules

Figure 2.2.1 shows a top level module hierarchy chart. It shows all the types of problems inwhich this program allows the user to conduct a drill. There are further breakdowns within each of these modules as discussed under top down development. Throughout the development of these modules, the author has adhered to these module characteristics:

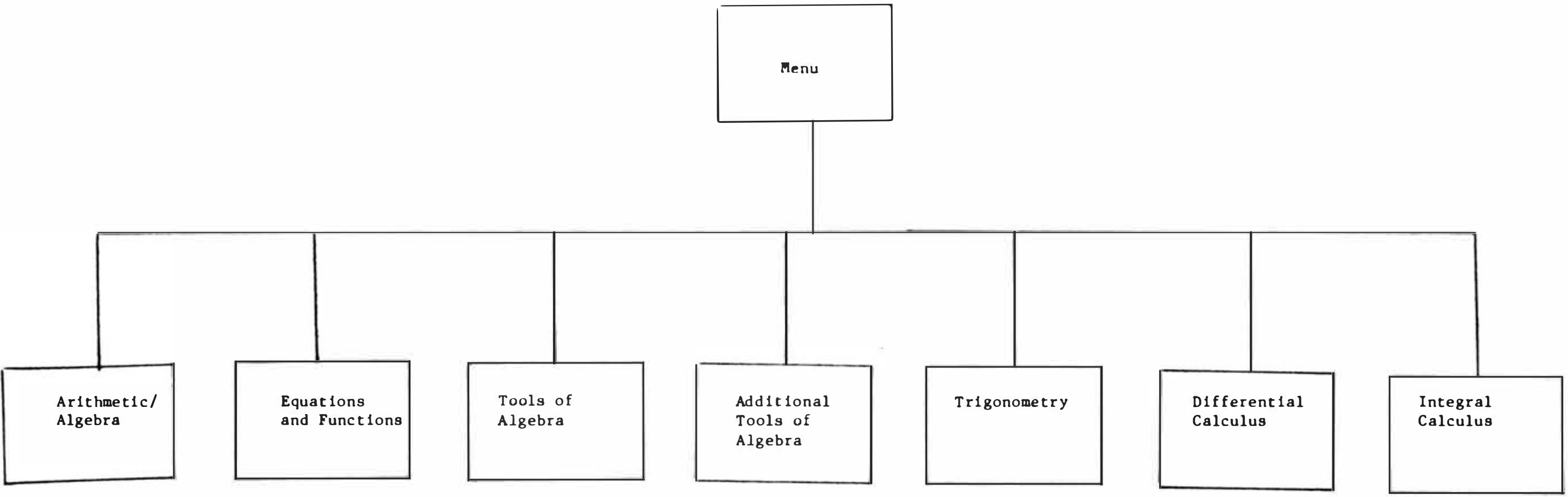
(a). single entry, and single exit via call and return mechanism.

(b). independent of "outside world". In other words, structure and other non-essential characteristics of data manipulated by a module should be "hidden" from other modules.

(c). each module is relatively small.

(d). each module performs a single function--each module solves only one problem type.

Figure 2.2.1


Menu
Arithmetic/
Algebra
Equations
and Functions
Tools of
Algebra
Additional
Tools of
Algebra
Trigonometry
Differential
Calculus
Integral
Calculus

(e). each module was compiled and tested separately to enhance thorough "white-box" testing. Due to language restrictions, each module had to be developed as a separate program to be tested for the function for which it was designed. Later on each of these modules were made subprograms (or procedures) to make an integrated whole.

2. Top Down Design

A top down design was used. The main module, or menu program, was developed before any other module. To test it, the seven modules displayed in Figure 2.2.1 were represented by program stubs. A program stub is a temporary module required to allow a partially completed program to execute. The actions of the stubs were limited to "Entered Stub X" messages, and return statements.

The most basic module is a function or a procedure. Each module was desinged to perform a single function. In actuality, there are primarily two types of modules, namely: those that allow for the selection of other modules, and those that actually solve particular problem types. For example, the menu module TUTORIAL, selects any of the seven problem types, namely: Arithmetic/Algebra, Equations and Functions, Tools of Algebra, Additional Tools of Algebra, Trigonometry, Differential Calculus and Integral Calculus. Each of these problem types may have further type breakdowns inwhich case they are to serve as module selectors. For example, Arithmetic/Algebra will serve to select the following modules: addition/subtraction of fractions, multiplying fractions, dividing fractions, and expanding expressions in general. These later modules are actual problem solvers. Differential Calculus, even though it is on the same logical level as the other seven modules called by the main module, is a problem solver and not a problem selector.

A separate subprogram was designed whenever the author determined that a section of code was going to be used several times with variations that could easily be effected through actual/formal parameter transmission. Caution was exercised not to develop procedures (modules) that are extensively general and complicated. This will enhance ease of module modification should it be deemed necessary in the future.

In what follows: the function, the input requirements, the output, and at times, the theory and the process of each module are discussed.

3. Module Specification

TUTORIAL

Function: to allow one to solve problems in the following areas:

Arithmetic/Algebra
Equations and Functions
Tools of Algebra
Additional Tools of Algebra
Trigonometry
Differential Calculus
Integral Calculus-the indefinite integral.

Input: Any integer in the range of 1 to 7 would be a valid input. Anything else must be flagged and the user given two additional chances to make a valid selection before the session is automatically terminated.

Output: The result of this module is to let the user into the desired problem area, and therefore, it has no tangible output.

Figure See Figure 2.2.1.

ARITHMETIC/ALGEBRA:

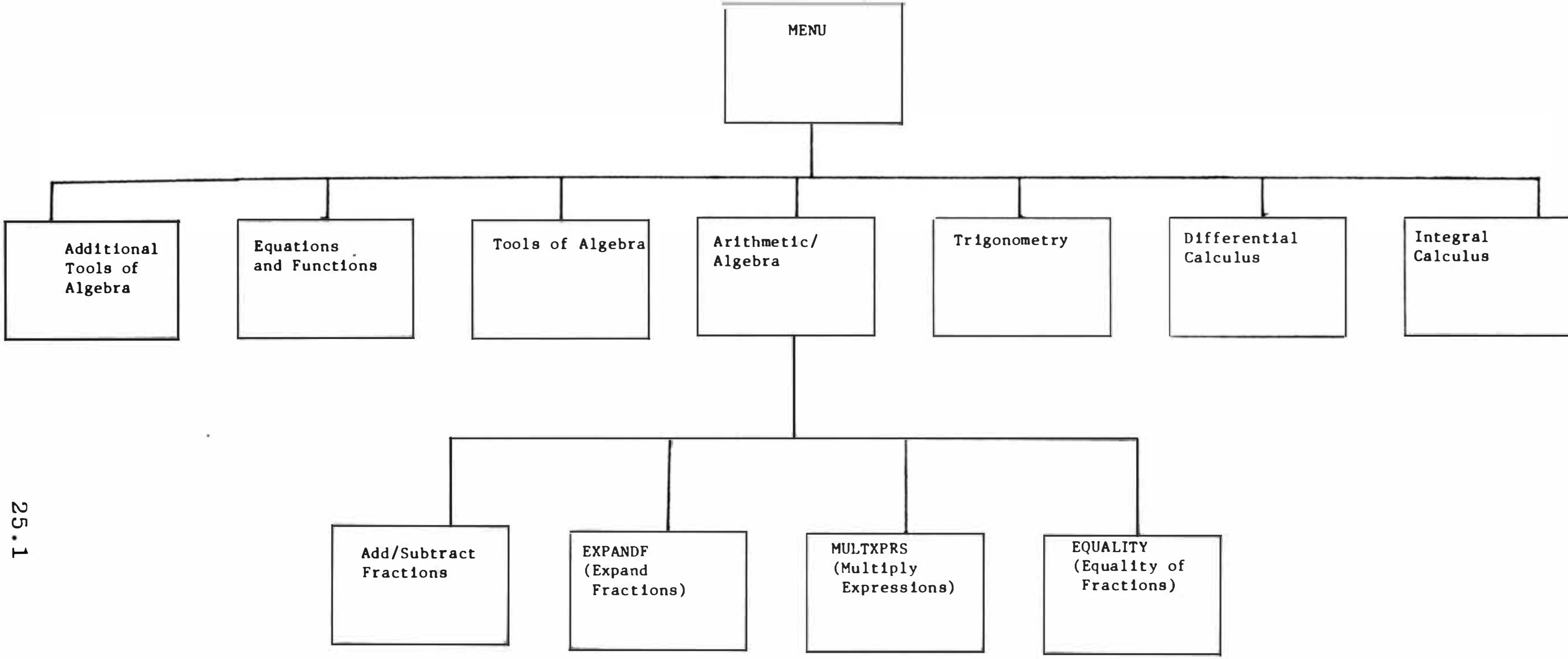
Function: to allow for the selection of the following problem types:

*addition/subtraction of fractions,
*expansion of algebraic expressions,
*multiplication of fractions (arithmetic as well as algebraic),
*determine whether or not two fractions are equal.

Input: Any integer in the range of 1 to 4 would be a valid input. Anything outside this range will be flagged and the user provided two additional chances to make a valid selection.

Output: The result of a valid selection will place the user in the desired environment—an environment that would allow the user to practice solving problems of the type selected.

Figure See Figure 2.2.2.


MENU
Additional Tools of Algebra
Equations and Functions
Tools of Algebra
Arithmetic/ Algebra
Trigonometry
Differential Calculus
Integral Calculus
Add/Subtract Fractions
EXPANDF (Expand Fractions)
MULTXPRS (Multiply Expressions)
EQUALITY (Equality of Fractions)

Arithmetic/Algebra was selected in the main module. This module will allow for the selection of any of the following modules: ADDSUBFR, EXPANDF, MULTXPRS, and EQUALITY. Even though the module READER is not included above, it is also used by this module, Arithmetic/Algebra, for input purposes. It has been developed as a sister to all the modules in the same level as Arithmetic/Algebra. A brief account of each of these modules is given below:

READER

* reads the entire 80-column record,
* eliminates all blanks from the string just read, and
* returns the string to the calling routine via formal/actual parameters.

ADDSUBFR

* displays prompts for the user explaining what the module expects,
* reads the information (problem) provided by the user via the READER module,
* displays the problem just read,
* uses FORMAC facilities to convert the PL/I character string into a mathematical expression for evaluation,
* determines the lowest common denominator, and then evaluates the expression,
* displays the lowest common denominator, and the resulting fraction after combination,
* the process is repeated until the command "/*" is issued. Upon termination of this module, control is returned to the program (module) Arithmetic/Algebra.

EXPANDF

A fraction is said to have been expanded to higher terms when upon application of the fundamental principle results in a fraction having a larger numerator and a larger denominator (12). This module will assist in solving problems inwhich a fraction and the desired denominator is given. The program will determine the term with which to multiply the numerator and the denominator to get an expanded fraction. This module performs the following tasks:

* displays prompts for the user for the problem to be entered,
* reads the problem via READER procedure,
* displays the problem just read for verification,
* uses FORMAC facilities to effect transformations and simplifications,
* displays the term, and
* displays the new numerator and the new denominator.

This process is repeated until the command "/*" is issued inwhich case control is returned to the program (MODULE) Arithmetic/Algebra.

MULTXPRS

This module is dedicated to multiplying mathematical expressions. It uses both the multinomial and the distributive laws to expand all products and positive integral powers of sums occurring in the expressions. The following tasks are accomplished by this module:

* display prompts for the user to enter a problem,
* reads the problem entered,
* converts the character string into a FORMAC expression,
* perform the expansion on the FORMAC expression, and
* display the result.

This process is repeated until the QUIT command or the END command is issued. Either of these would return control to the Arithmetic/Algebra module.

EQUALITY

Two fractions are said to be equal if and only if their cross products are equal. This module is dedicated to determining whether or not two fractions are equivalent. The following tasks are performed by this module:

* displays a prompt telling the user what input is expected and in what format,
* the two fractions are read via READER module separated by a comma,
* converts the two fractions into FORMAC expressions,
* extracts the numerators and the denominators of both fractions using FORMAC73 facilities,
* calculates the left-hand side (LHS) by cross multiplication of the numerator of the first fraction by the denominator of the second fraction,
* calculates the right-hand side (RHS) by cross multiplication of the numerator of the second fraction by the denominator of the first fraction, and
* compares the two--the LHS and the RHS and displays a message as to whether these fractions are equivalent.

EQUATIONS AND FUNCTIONS:

Function: to allow for the selection of the following types of problems:

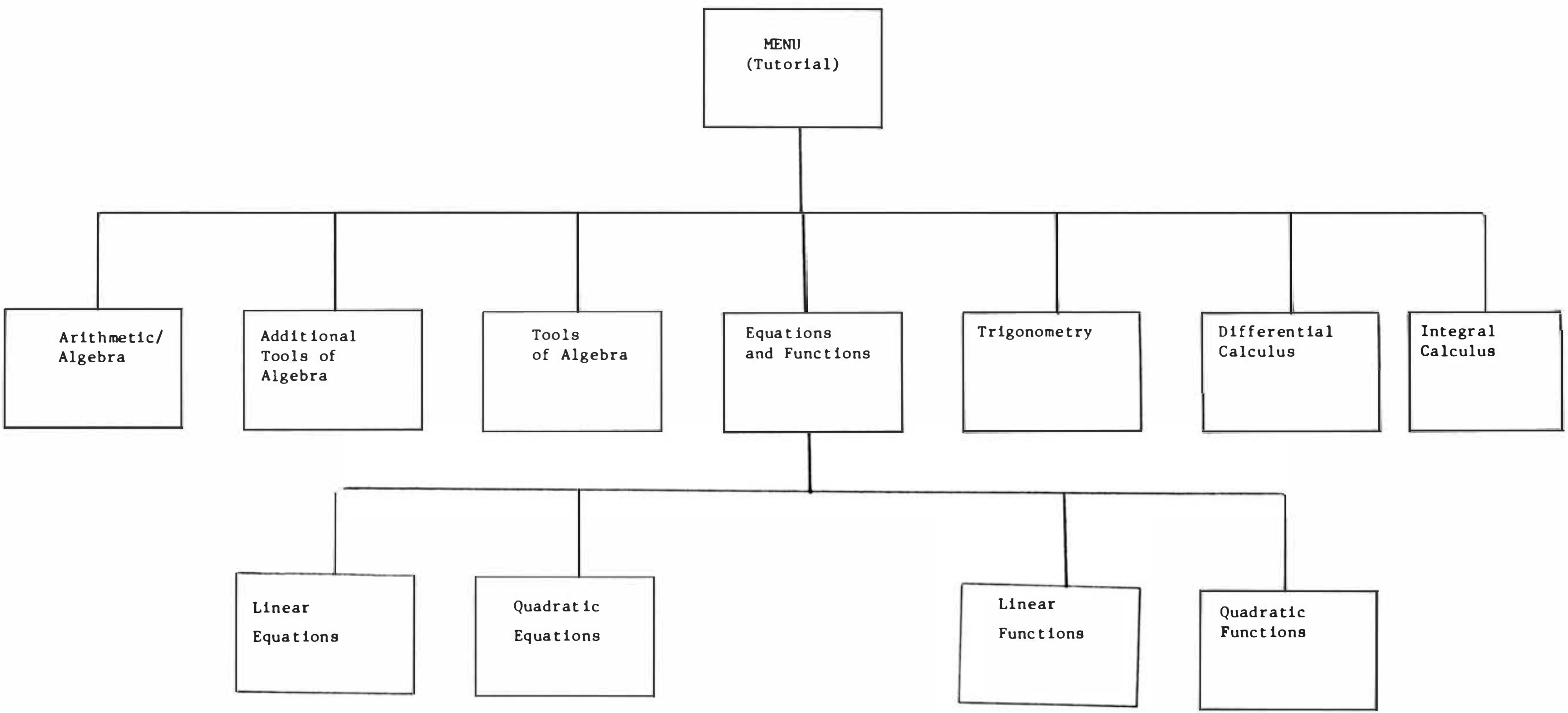
solution sets of linear equations
solution sets of quadratic equations
problems involving the linear function
problems involving the quadratic function

Input: Any integer in the range of 1 to 4 would constitute a valid input. Anything outside this boundary will be flagged and the user will be given two more chances to enter a valid selection.

Output: The result of a valid selection will place the user in the desired environment--one that would allow him/her to practice solving problems in that area. Once in the selected environment, the input would be dictated by the nature of the problem to be solved.

Figure: See Figure 2.2.3.

Figure 2.2.3


MENU
(Tutorial)
Arithmetic/ Algebra
Additional Tools of Algebra
Tools of Algebra
Equations and Functions
Trigonometry
Differential Calculus
Integral Calculus
Linear Equations
Quadratic Equations
Linear Functions
Quadratic Functions

Solution Sets of Linear Equations:

Function: This module implements most of the material in Unit 9 of Intermediate Algebra by Joseph Newmeyer, Jr., and Gus Klentos (12). Two types of problems are implemented, namely:

solving linear equations
absolute value equations

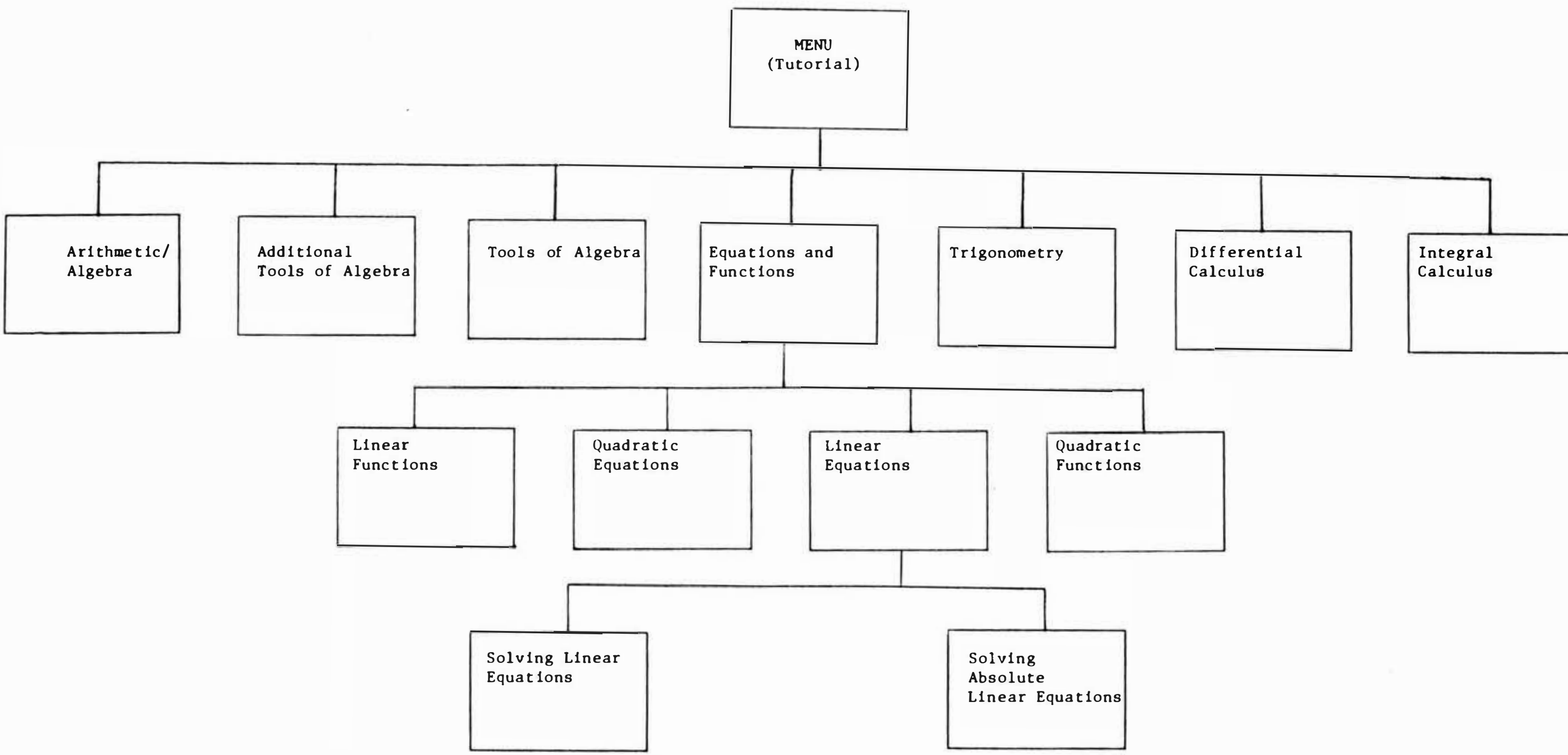
It will serve to allow for the selection of one of these problems.

Input: Any entry of either 1 or 2 would be a valid input. Anything else woul be flagged and the user given two more chances to make a vali selection.

Output: The result of a valid selection will place the user in the appropriat environment. The input there would be dictated by the type of th problem to be solved.

Figure: See Figure 2.2.4.

Figure 2.2.4


MENU
(Tutorial)
Arithmetic/ Algebra
Additional Tools of Algebra
Tools of Algebra
Equations and Functions
Trigonometry
Differential Calculus
Integral Calculus
Linear Functions
Quadratic Equations
Linear Equations
Quadratic Functions
Solving Linear Equations
Solving Absolute Linear Equations

Solving Linear Equations:

Function: This module will find a solution set of a given linear equation over the set of real numbers.

Input: The input to the module will be any linear equations. The equation may appear on an 80-byte record.

Output: The solution set of the given linear equation.

A linear equation is an equation which is equivalent to an equation of the form $ax = b$, $a, b \in R$, $a \neq 0$. Example of linear equations include (12: pp. 96):

$3x = 5,$

$7x - 2 = 8,$

$x = 0,$ and

$5x - 1 = x + 3$

Example 1

Find the solution set over R of $6x - 1 = 5$.

$$6x - 1 + 1 = 5 + 1$$

$$6x = 6$$

$$\frac{6x}{6} = \frac{6}{6}$$

$$x = 1$$

The solution set is $\{1\}$

SYSTEMS1

This module is dedicated to solving linear equations for a given unknown. It performs the following tasks:

* displays appropriate prompts to the user.
* reads the input linear equation to be solved via READER Module. The linear equation is separated from the unknown variable to be solved for via a comma.
* breaks the equation into two parts, namely: the left hand side (LHS), and the right hand side (RHS).
* it tests both sides for parenthesizes expressions and then expands them out if they exist.
* combines the like terms.
* moves all but the terms containing the variable to be solved for onto the right hand side.
* extracts the coefficient of the term on the left hand side.
* divides both sides by this coefficient to yield the desired solution.
* this process is continued as long as there are more linear equations to solve or until the command "/*" is issued.

Once this module is executed, control is returned to the module for solving linear equations.

Absolute

This module will solve absolute linear equations for a given unknown. The majority of the tasks have been solved by two SYSTEMS1 Modules. It will call twice--the first one with the RHS as is, and the second time with negative RHS. Each time the same tasks performed for linear equations (SYSTEMS1) are performed.

Absolute Value Equations:

Function: to solve absolute value equations.

Input: The equation may appear anywhere on an 80-byte record.

Output: The solution set of the given absolute equation.

We know that $|3| = 3$, $|0| = 0$, $|-2| = 2$. The absolute value of a real number is always non-negative. We cannot say that $|x|$ equals x because x may be negative (12: pp.98).

$$\text{Definition} |x| = \begin{cases} x \text{ if } x \quad 0 \\ 0 \text{ if } x = 0 \\ -x \text{ if } x \quad 0 \end{cases}$$

Example

Suppose $|x - 2| = 5$. This means that $(x - 2)$ is either 5 or -5 since $|5| = 5$ and $|-5| = 5$.

$$x - 2 = 5, \text{ or } x - 2 = -5$$
$$x = 7 \qquad x = -3$$

Check $x = -3$: $|(-3) - 2| = 5$

$|-5| = 5$

$\underline{5 = 5}$

Check $x = 7$: $|7 - 2| = 5$

$|5| = 5$

$\underline{5 = 5}$

The solution set $\{-3, 7\}$.

Solution sets of quadratic equations:

Function: to solve selected problems related to quadratic equations, namely: finding the roots of the quadratic equations and determining the possible quadratic equation given a solution set. The modules that will accomplish this function are broken down as follows:

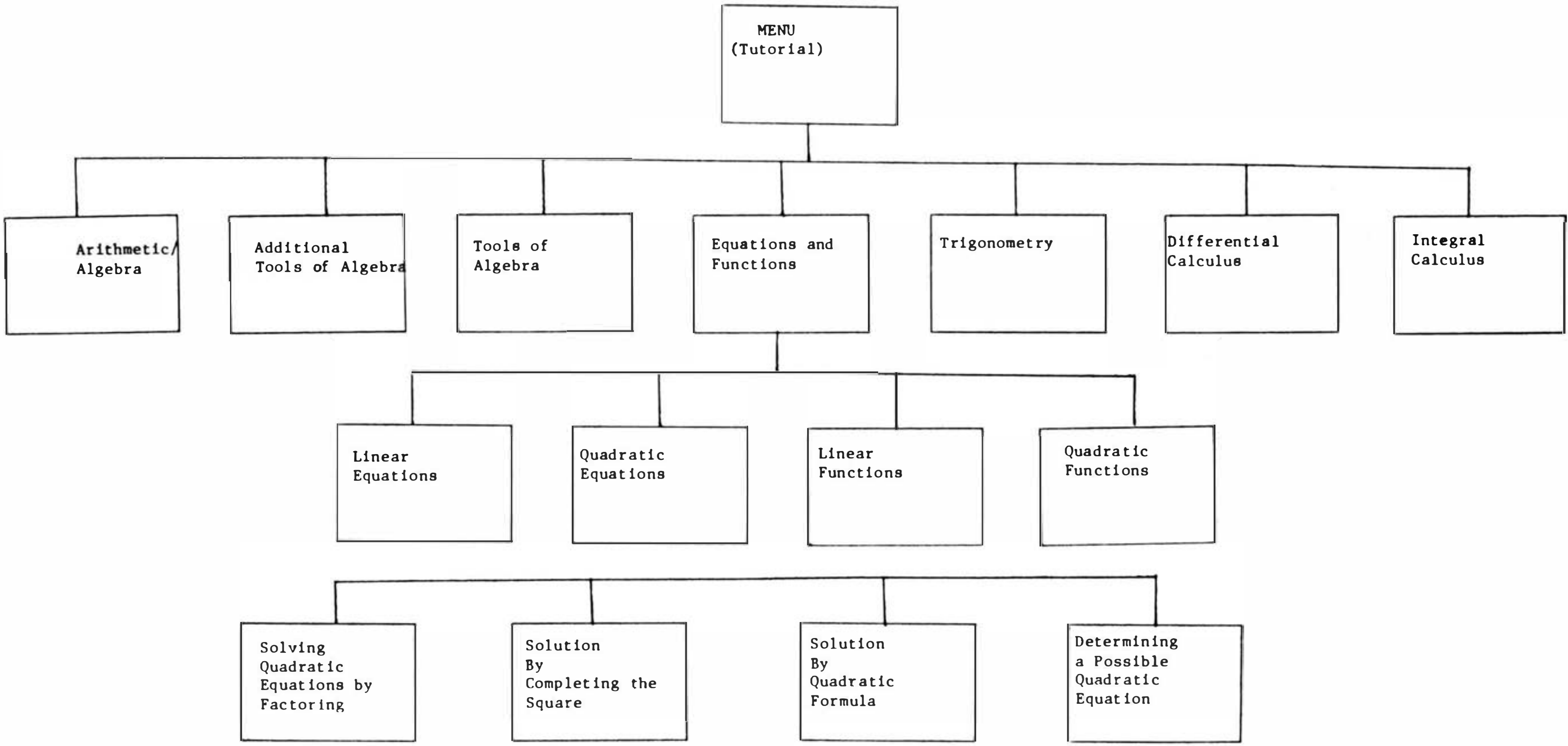
solving a quadratic equation by factoring
solving a quadratic equation by completing the square
solving a quadratic equation by the quadratic formula
determining a possible quadratic equation given a solution set

Input: Any integer number ranging between 1 and 4 inclusively would constitute valid input. Anything else will be flagged as invalid, and the user given two more chances to enter a valid selection.

Output: The product of this module is that the user will enter the environment for solving the selected problem type.

Figure: See Figure 2.2.5.

Figure 2.2.5


MENU
(Tutorial)
Arithmetic/
Algebra
Additional
Tools of Algebra
Tools of
Algebra
Equations and
Functions
Trigonometry
Differential
Calculus
Integral
Calculus
Linear
Equations
Quadratic
Equations
Linear
Functions
Quadratic
Functions
Solving
Quadratic
Equations by
Factoring
Solution
By
Completing the
Square
Solution
By
Quadratic
Formula
Determining
a Possible
Quadratic
Equation

Solving a quadratic equation by factoring:

Function: to factor quadratic equations that have real factors. It will reject those quadratic equations that have imaginary roots, or those which do not factor into real numbers. Once the equation is factored, the roots may be readily obtained.

Input: A quadratic equation separated by a comma fromt the variable constituting the equation. This entry may appear anywhere on an 80-byte record image.

Output: The quadratic equation in its factored form and the solution.

Example

$-x + x^2 - 12 = 0$

$x^2 - x - 12 = 0$

Results in:

$(x + 3)(x - 4)$

$x = -3$

$\underline{x = 4}$

Solving quadratic equations requires an understanding of two basic theorems related to the properties of zero (12: pp. 106).

Theorem 1

a, $b \in R$. If $a = 0$ or $b = 0$ then $a * b = 0$

Theorem 2

$a, b \in R$. If $a * b = 0$, then $a = 0$ or $b = 0$

The product of two factors equals zero if and only if one or both of the factors equals zero. $a * b = 0$ if and only if $a = 0$ or $b = 0$. The word "or" is used in the inclusive sense.

Example: Find the solution set of $x^2 + 2x - 15 = 0$ by factoring.

$x^2 + 2x - 15 = 0$

$(x + 5)(x - 3) = 0$

If the product of two numbers is zero, one or both of the numbers is zero.

$(x + 5) = 0$ or $(x - 3) = 0$

$x = -5$ or $x = 3$

The solution set is (3, -5).

Finding Factors:

In finding factors of a trinomial expression, one comes up with a list of all possible factorizations, and then by the process of elimination only the correct factor will be left. Through experience one is often able to eliminate certain possibilities more readily, thus leaving fewer possibilities to consider.

For example, factor $6x^2 + 13xy - 5y$.

Solution: 6 is the product of 6 and 1 or 2 and 3.

5 is the product of 1 and 5.

Possible factorizations include:

$(3x + 5y)(2x - y)$,

$(3x - 5y)(2x + y)$,

$(3x + y)(2x - 5y)$ and

$(3x - y)(2x - 5y)$.

Correct factorization is $(3x - y)(2x - 5y)$.

An alternate way to effect factorization is going about it backwards. It is sometimes easier to obtain the factors through other means, such as the quadratic formula, and then come up with a factored expression. This is particularly applicable in cases where there are so many possibilities, and one detects the possibility of not having any real factors.

When this alternate method is adopted, then one must be prepared to accept a correct answer in a different form, or perform further manipulations to get the factored expression in the form that it would have been in had the method of elimination been adopted. For example, let us reconsider the following trinomial expression:

$$6x^2 + 13xy + 5y^2$$

By using the quadratic formula, we get the following:

$$x = \frac{-13y \pm \sqrt{(13y)^2 - (4)(6)(5y^2)}}{(2)(6)}$$

$$= \frac{-13y \pm 7y}{12}$$

$$x = (-1/2y, -5/3y)$$

the factored form becomes: $(x + 5/3y)(x + 1/2y)$.

$(x - 1/3y)(x + 5/2y)$ is mathematically equivalent to $(3x - y)(2x + 5y)$, which was the correct factor among the rest of the possible factors. And indeed, one can get the factored expression arrived at by the alternate method by the following manipulations:

$(x - 1/3y)$ when multiplied by 3 yields $(3x - y)$

$(x + 5/2y)$ when by yields $(2x + 5y)$, and hence $(3x - y)(2x + 5y)$.

Process

In order to solve such a problem (factorization) on the computer, one would need to come up with a procedure to generate all the possible factors and then either select the correct one, or prove that there are no real factors by the process of elimination. This will, unfortunately, be limited to quadratic equations with numeric constants as the coefficients. Such a procedure can be enhanced to accommodate non-numeric algebraic constants as well. Alternately, some of the most sophisticated factoring algorithms may be comployed. These are deemed by the author to be beyond the scope of this paper. Instead, a simpler and a representative alternative is adopted, and the repercussions well noted.

In this paper, the alternative method described above will be adopted. This consequently implies that in some cases, the factored form of some quadratic equations will be different from the form that would otherwise be obtained by the first method. For example: factor $6x^2 + 13xy + 5y$ yields the following:

$$(x + 0 \cdot 3333333y)\ (x + 2 \cdot 5y)$$

Completing the square of a quadratic equation:

Function: to complete the square of a quadratic polynomial of the form

$x_2 + k * x.$
$x^2 + k * x$

Input: Any quadratic polynomial in the form x + k * x would be valid input.

Output: The binomial form of the expression square i.e. $(x + k)^2$ which when expanded yields the perfect square trinomial.

Example

$$x^2 + 6x \text{ yields } x^2 + 6x + 9 = (x + 3)^2$$

Some quadratics are perfect square trinomials. For example, $x^2 + 10x + 25 = (x + 5)^2$. Since $(x + a)^2 = x^2 + 2ax + a^2$, the last term a^2 is the square of one-half the coefficient of the x term. That is $a^2 = (\frac{1}{2}(2a))^2$ (12: pp. 110).

A quadratic polynomial of the form x + k * x can be transformed into a perfect square trinomial by adding the square of one-half the coefficient of x. The process is known as completing the square.

Solving a quadratic equation by completing the square is how the quadratic formula is arrived at. In fact, the quadratic formula is the general case for solving a quadratic equation by completing the square.

Example 1

Solve: $x^2 - 4x - 6 = 0$

$$x^2 - 4x = 6$$

$$x^2 - 4x + 4 = 6 + 4$$

$$(x - 2)^2 = 10$$

$$x - 2 = \underline{+}\sqrt{10}$$

$$x - 2 = -\sqrt{10}, \quad x - 2 + \sqrt{10}$$

$$x - 2 = -\sqrt{10}, \; x = 2 - \sqrt{10}$$

The solution set is $\{2 + \sqrt{10}, \quad 2 - \sqrt{10}\}$

Example 2

Complete the square on: $ax^2 + bx + c = 0$, $a = 0$ yields the following:

$$ax^2 + bx + c = 0$$

$$x^2 + \frac{b}{a}x = -\frac{c}{a}$$

$$x^2 + \frac{b}{a}x + \frac{b^2}{4a^2} = \frac{b^2}{4a^2} - \frac{c}{a}, \qquad \frac{1}{2} \cdot \frac{b}{a}^2 = \frac{b^2}{4a^2}$$

$$x^2 + \frac{b}{a}x + \frac{b^2}{4a^2} = \frac{b^2}{4a^2} - \frac{4ac}{4a^2}$$

$$x + \frac{b}{2a}^2 = \frac{b^2 - 4ac}{4a^2}$$

$$x + \frac{b}{2a} = +\sqrt{\frac{b^2 - 4ac}{4a^2}}, \qquad x + \frac{b}{2a} = -\sqrt{\frac{b^2 - 4ac}{4a^2}}$$

$$x = -\frac{b}{2a} + \sqrt{\frac{b^2 - 4ac}{4a^2}}, \qquad x = -\frac{b}{2a} - \sqrt{\frac{b^2 - 4ac}{4a^2}}$$

Case 1. $a > 0$ (then $a^2 = a$)

$$x = \frac{-b}{2a} + \sqrt{\frac{b^2 - 4ac}{-2a}}, \qquad x = \frac{-b}{2a} - \sqrt{\frac{b^2 - 4ac}{-2a}}$$

The solution set is $\quad \dfrac{-b + \sqrt{b^2 - 4ac}}{2a}, \qquad \dfrac{-b - \sqrt{b^2 - 4ac}}{2a}$

Case 2. $a < 0$ (then $a^2 = -a$)

$$x = \frac{-b}{2a} + \frac{b^2 - 4ac}{2a}, \qquad x = \frac{-b}{2a} - \frac{b^2 - 4ac}{2a}$$

The solution set is $\quad \dfrac{-b - \sqrt{b^2 - 4ac}}{2a}, \qquad \dfrac{-b + \sqrt{b^2 - 4ac}}{2a}$

In either case, $a < 0$ or $a > 0$, the solution set is:

$$\left\{ \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \qquad \frac{-b - \sqrt{b^2 - 4ac}}{2a} \right\}$$

Solving a Quadratic Equation By the Quadratic Formula:

Function: to find the real roots of a quadratic equation if they exist by using the quadratic formula.

Input: A quadratic equation separated from the variable used in the equation by a comma. For example $x^2 - 5x - 6, x$.

Output: The roots of the given quadratic equation if they exist, otherwise, pertinent messages such as the following are printed:

> this is not a quadratic equation
> there are no real roots, and so no further simplification will be attempted.

The general form of a quadratic equation is $ax^2 + bx + c = 0$, $a \neq 0$. A complete derivation of the quadratic formula is given in the previous section under, "Completing the Square of a Quadratic Polynomial".

Determining a Possible Quadratic Given a Solution Set:

Function: to determine a possible quadratic equation from a given solution set.

Input: the solution set separated by a comma.

Output: a possible quadratic equation in 'x'. The variable x was arbitrarily chosen.

The idea behind determining a possible quadratic equation from a given solution set is based on the theories under the section titled, "Determining the Factors of a Quadratic Equation". These were Theorem 1, and Theorem 2.

In this section, two problems will be solved to illustrate the basic steps:

<u>Example 1:</u> Find a possible solution set given (2,4) as a solution set.

The solution set is (2,4).

$x = 2$, or $x = 4$

$(x - 2) = 0$ or $(x - 4) = 0$

$(x - 2)(x - 4) = 0$

$x^2 - 6x + 8 = 0$

<u>Example 2:</u> Find two possible solution sets given (-1/3, 2) as a solution set.

The solution set is (-1/3, 2).

$x = -1/3$ or $x + 2$

$(x + 1/3) = 0$ or $(x = 2)$

$(x + 1/3)(x - 2) = 0$

$x^2 - 5/3x - 2/3 = 0$

Alternately:

The solution set is -1/3, 2

$x = -1/3$ or $x = 2$

$3x = -1$ or $x = 2$

$3x + 1 = 0$ or $(x - 2) = 0$

$(3x + 1)(x - 2) = 0$

$3x^2 - 5x - 2 = 0$

THE LINEAR FUNCTION:

Function: to facilitate the selection of the following problem solving modules:

intercepts of linear equations.
slope of a function when given two points.
determine the equation of a line given:

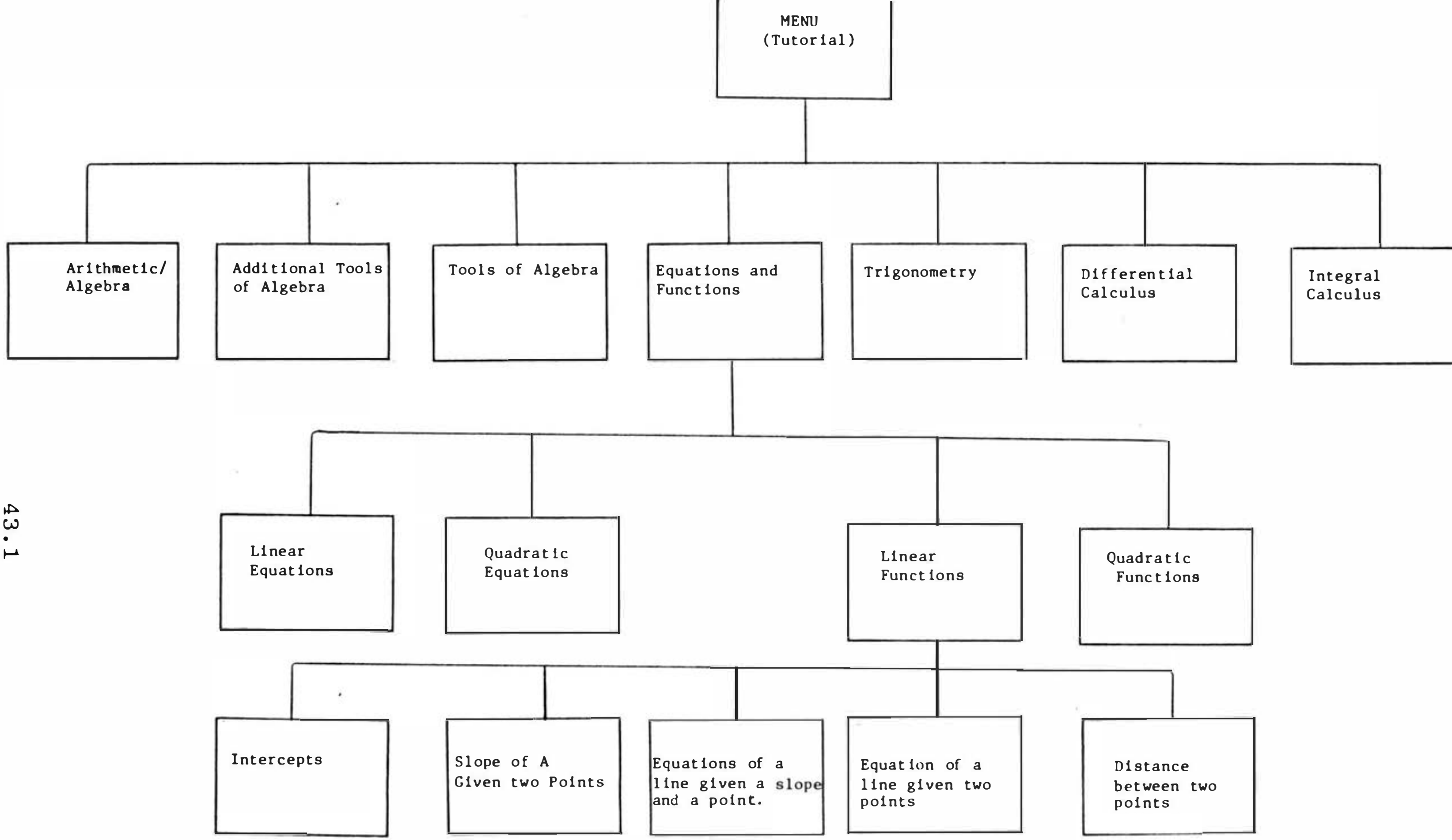
(a) the slope of the line, and
(b) a point through which the line passes

determine the equation of a line given two points (a,b), (c,d)
calculate the distance between two points by using the pythagorean theorem.

Input: Any number between 1 and 5, inclusively. Any number outside this range will be flagged as invalid, and the user will be given two more chances to make the right selection.

Output: The result of a valid selection would place the user in the desired environment--an environment that would allow one to practice solving the problems of the selected type.

Figure: See Figure 2.2.6.


MENU
(Tutorial)
Arithmetic/ Algebra
Additional Tools of Algebra
Tools of Algebra
Equations and Functions
Trigonometry
Differential Calculus
Integral Calculus
Linear Equations
Quadratic Equations
Linear Functions
Quadratic Functions
Intercepts
Slope of A Given two Points
Equations of a line given a slope and a point.
Equation of a line given two points
Distance between two points

Intercepts of Linear Equations:

Function: to calculate x- and y-intercepts of a linear function.

Input: any linear function in x and y.

Output: the x- and y-intercepts of the given linear function.

The function defined by ax + by + c = 0 with a, b, c ∈ R, b = 0 is called a linear function since the graph of this function over RxR is always a straight line. Vertical lines do not represent functions. All non-vertical lines do represent functions. In order to sketch the graph of a line, a minimum of two points are required.

x- and y-intercepts

If a line is neither vertical nor horizontal, it will have one point in common with the y-axis. These points will be called the x-intercept and y-intercept respectively. The x-intercept point always has its second coordinate equal to zero. The y-intercept point always has its first coordinate equal to zero.

Finding x- and y- Intercepts

Thus to find the x-intercept, set $y = 0$ and solve the resulting equation for x. The point represented by $(x, 0)$ will be the x-intercept. To find the y-intercept, set $x = 0$ and solve the resulting equation for y. The point $(0, y)$ will be the y-intercept.

Example:

Find the x- and y-intercepts of $2x + y = 8$.

Solution: to find the x-intercept, set $y = 0$.

$2x + 0 = 8$

$2x = 8$

$\underline{x = 4}$

The x-intercept is $(4, 0)$.

To find the y-intercept, set $x = 0$.

$2(0) + y = 8$

$0 + y = 8$

$\underline{y = 8}$

The y-intercept is $(0, 8)$.

Slope of a Function When Given Two Points:

Function: to calculate the slope of a function from two points.

Input: two points where the first pair is separated from the second by a semi-colon, and the x in each pair is separated from the y by a comma. For example: x1, y1; x2, y2.

Output: the slope of the function, m.

The slope of a function is defined as the change in the y direction divided by the change in the x direction. Mathematically, this is written as:

$$m = \frac{\Delta y}{\Delta x}$$

where $\Delta x = x_2 - x_1$ (the change in x)
$\Delta y = y_2 - y_1$ (the change in y direction)

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$$

Example: (3, 2) and (5, 6)

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - y_1}$$

$$= \frac{(6 - 2)}{(5 - 3)} = \frac{4}{2}$$

$m = 2$

The slope of the line and the point (an ordered pair). The slope is separated from the point by a semi-colon.

Determine the equation of a line given the slope of a line, and a point through which the line passes:

Function: to determine the equation of such a line.

Input: the slope of the line, and a point through which the line psses . The x and y which constitute the pair are separated from each other by a comma. For example: m; x, y.

Output: the equation of the line passing through the given point, and having the given slope.

Example: Write the equation of a line through (2, 4) with slope 3.

Step 1: $m = 3, (x_1, y_1) = (2, 4)$

Step 2: $y - y1 = m(x - x1)$

Step 3: $y - 4 = 3(x - 2)$

Step 4: $y - 4 = 3x - 6$

Step 5: $-3x + y + 2 = 0$

Determine the equation of a line given two points:

Function: to determine an equation of a line when given two points.

Input: two ordered pairs of points. The first pair is separated from the next pair by a semi-colon. The x and the y members of each pair are separated by a comma.

Output: an echo-check of the ordered pairs entered and the resulting equation.

Process:

The processing is quite straight forward and goes through the following steps:

Step 1: the slope is determined from the two points:

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - y_2}$$

Step 2: One of the pairs of points is selected. Since it is immaterial as to which of the points is selected, the program has been prepared to always select the first one.

Step 3: Use the point slope formula along with the necessary simplifications to determine the equation of the line.

$$y - y_1 = m(x - x_1)$$

The distance between two points:

Function: to find the length of the line segment joining two points.

Input: two points separated by a semicolon. The x and the y components of each ordered pair are separated by a comma. For example: $x_1, y_1;$ x_2, y_2.

Output: the distance between the two points.

The following distance formula is used in calculating the distance between two points:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

SLOPEYIN

* Displays the prompts for the user to enter the two points characterizing the line whose slope is to be determined.

* Reads the two points as characters via another module, READER.

* Displays the two points after some processing.

* Calculates delta x.

* Calculated delta y.

* Displays delta x and delta y.

* Calculates the slope.

* Displays the slope.

* Repeats this process from the beginning until "/*" has been issued, at which point control is returned to the calling module. The following return of control applies to this module: It returns control to LINEARFN (linear functions), LINEARFN returns control to EONSFUNCS (equations and functions), and EONFUNCS returns control to the top level module TUTORIAL.

EQNPTSLP

* Displays the prompts for the user to enter the point, and the slope from which the equation of the line is to be derived.

* Reads the point, and the slope as character strings via another module called READER.

* The point, and the slope are displayed after minor processing.

* The equation of the line is formulated using the point slope formula.

* The equation of the line is displayed.

* This process is repeated from the beginning until "/*" has been issued, at which point control is relinquished from this module. The following return sequence is followed: Control is returned to LINEARFN (linear functions), LINEARFN returns control to EQNFUNCS, EQNFUNCS returns control to the top level module, TUTORIAL.

DISTANCE

* Displays the prompts for the user to enter two points from which the program will calculate distance in between them.

* The two points are read as character strings via another module, READER.

* The two points are displayed after some pre-processing.

* The two points are used in the distance between two points formula to calculate the distance.

* The distance between the two points is displayed.

* This process is repeated from the beginning until "/*" is issued, at which point control is returned to the calling module. The following control return sequence is executed: DISTANCE returns control to LINEARFN, LINEARFN returns control to EONFUNCS which in turn returns control to the top level module, TUTORIAL.

XYINTCPT

* Displays the prompt for the user to enter the function whose x-, and y- intercepts are to be calculated.

* Reads the functions as a character string via another module called READER.

* Displays the function.

* Converts the function as a character string into a FORMAC73 format (or expression) amenable to mathematical manipulation.

* Sets the value of y to zero.

* Calculates the x- intercept.

* Displays the x- intercept.

* Sets x to the value of zero.

* Calculates the y- intercept.

* Displays the y- intercept.

* Repeats this process from the beginning until "/*" has been issued, at which point control is returned to the calling module. The calling module in this case was LINEARFN (linear functions) which was called by another module EQNFUNCS (equation and functions), which was called by the main module (TUTORIAL).

THE QUADRATIC FUNCTION

Function: to allow for the selection of the modules dedicated to manipulating or solving problems with the quadratic function, namely:

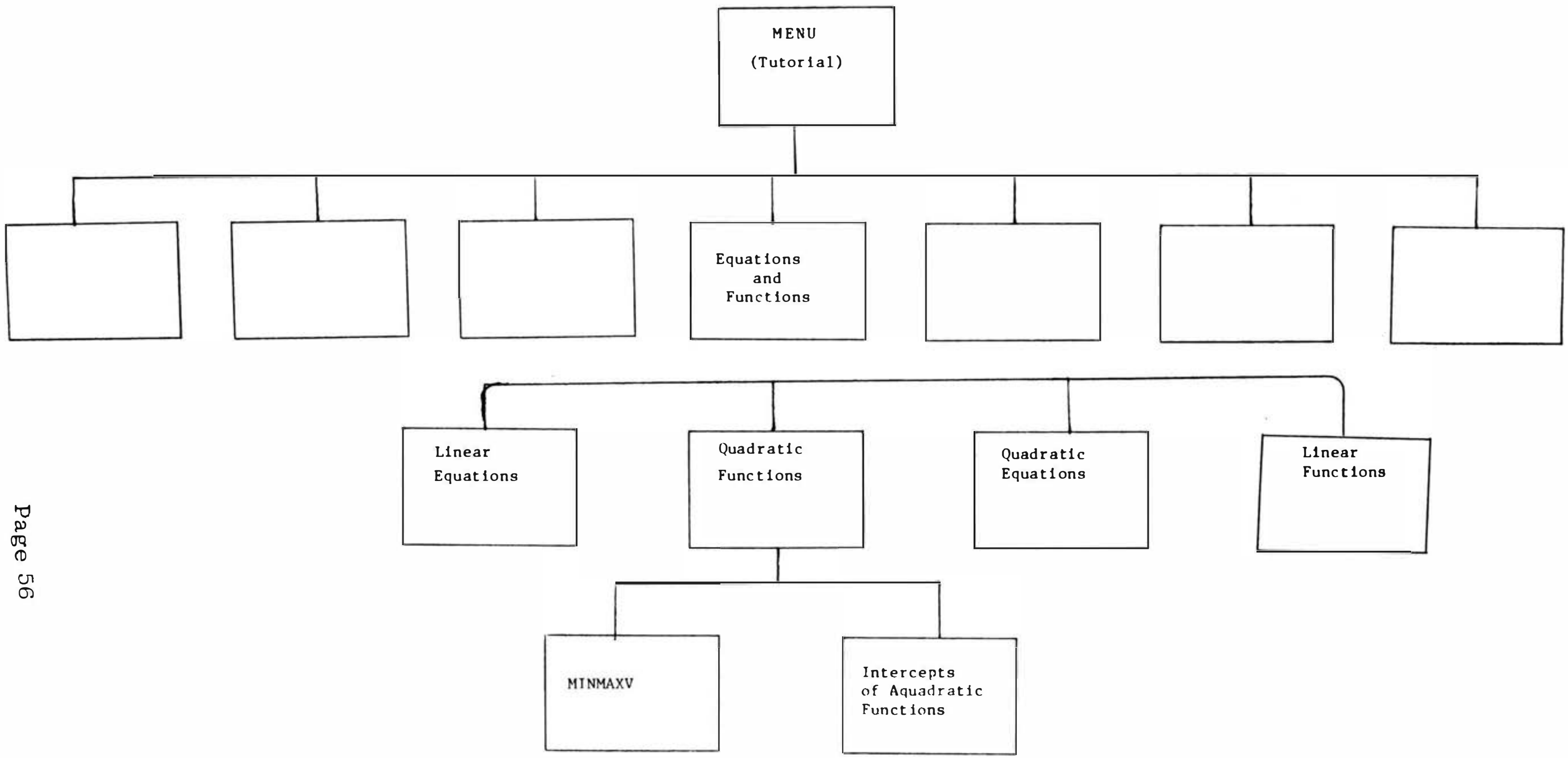
determining the intercepts of a quadratic function.
determining the vertex, minimum/maximum value of a quadratic function.

Input: Either 1 or 2 would be a valid selection. Anything else will be flagged, and the user given two more chances to enter a valid selection.

Output: The result of a valid selection will put the user in the appropriate environment--an environment that would allow the user to practice solving problems of the selected type.

Figure: See Figure 2.2.7.

Figure 2.2.7


MENU
(Tutorial)
Equations and Functions
Linear Equations
Quadratic Functions
Quadratic Equations
Linear Functions
MINMAXV
Intercepts of Aquadratic Functions

Determining the Vertex, the Maximum/Minimum Value of a Ouadratic Function:

Function: to determine whether the function has a maximum or a minimum.
to calculate the vertex.
to calculate the maximum or the minimum.

Input: a valid quadratic function.

Output: identification as to whether the function has a maximum of a minimum.

the vertex of the function, and
the extremum (either the maximum, or the minimum value as previously identified).

If x is any real number, the square of x is never negative. That is, if $x \in R$, then $x^2 > 0$. When a real number is squared, the smallest possible result is zero.

The maximum or minimum of a parabola

Given $y = ax^2 + bx + c$, $a, b, c \in R$, $a \neq 0$

1. The graph is a parabola
2. If $a > 0$, the parabola has a minimum
3. If $a < 0$, the parabola has a maximum

Finding minimum or maximum

An example will be used to illustrate the steps involved. They are primarily based on the principles guiding solving a quadratic equations by completing the square.

Example Find the minimum or the maximum of $y = x^2 + 6x + 10$.

Step 1 $y = x^2 + 6x + 10$

Step 2 $y = x^2 + 6x + 9 + 10 + (-9)$ complete the square

Step 3 $y = (x + 3)^2 + 1$

The square of any number is greater than or equal to zero. The lowest value $(x + 3)^2$ can take is zero; thus the lowest value y can take is +1.

The minimum is 1; it occurs when $(x + 3)^2$ becomes zero or when $x = -3$.

The vertex of the porabola is (-3, 1).

QUADFUNC

* Displays the prompts for the user to enter the quadratic functin whose x-, and y-intercepts are to be calculated.
* Reads the quadratic function as a character string via another module, READER.
* Displays the quadratic function as a character string after some pre-processing.
* Breaks down the function into two components: the left-hand side (lhs) and the right-hand side (rhs).
* Converts the rhs into a FORMAC73 expression since the lhs has only "y".
* Sets "y" to the value of zero.
* Calculates the x- intercepts using the quadratic formula.
* Displays the x- intercepts.
* Sets "x" to the value of zero.
* Calculates the y- intercept.
* Displays the y- intercept.
* Repeats this process from the beginning until a "/*" is issued, at which point control is returned to the calling module. The following control return sequence is executed: QUADFUNC returns control to QUADFNC, QUADFNC returns control to EQNFUNCS, which returns control to the top level module, TUTORIAL.

MINMAXV

* Displays prompts for the user to enter the quadratic function whose minimum/or maximum, and whose vertex is to be calculated.
* Reades the function as a character string via another module, READER.
* Breaks down the function into two components: the left-hand side (lhs), and the right-hand side (rhs).
* Converts rhs into a FORMAC73 expression to facilitate mathematical manipulation.
* Completes the square on the rhs by an established procedure.
* Displays the rhs before the completing of the square begins.
* Determines the vertex after completing the square.
* Displays the vertex.
* Determines whether or not the function has a maximum or a minimum via the coefficient of x^2.
* Calculates the extremum (maximum or minimum) and then displays the results.
* Repeats this whole process from the beginning until a "/*" is issued, at which point control is returned to the calling module. The following control return sequence is executes: MINMAXV returns control to QUADFN, QUADFN returns control to EQNFUNCS which returns control to the top level module, TUTORIAL.

TOOLS OF ALGEBRA:

Function: to allow for the selection of the following types of problems:

Evaluate a 2x2 determinant.
Evaluate a 3x3 determinant.
Use Cramer's rule to solve problems with two linear equations, an two unknowns.
Use Cramer's rule to solve problems with three linear equations, and three unknowns.

Input: Any integer in the range of 1 to 4 would constitute a valid input. Anything outside this boundary will be flagged and the user given two additional chances to enter a valid selection.

Output: The result of a valid selection will place the user in the desired environment—one that would allow the user to practice solving a particular type of problem. Once in that environment, the input is dictated by the nature of the problem.

DETERM2

Function: to evaluate a 2x2 determinant.

Input: elements of the determinant.

Output: the second order determinant.

By definition:

$$\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix} = a_1b_2 - a_2b_1$$

Example:

$$\begin{vmatrix} 2 & 1 \\ 4 & 5 \end{vmatrix} = (2)(5) - (1)(4)$$

$$10 - 4$$

The determinant = 6.

DETERM3

Function: to evaluate a 3 x 3 determinant.

Input: elements of the determinant.

Output: the value of the determinant along with the intermediate steps.

A determinant with three rows and three columns is called a third order determinant. A third order determinant may be expanded as follows (102: pp. 275):

Step 1: Rewrite the first and the second columns to the right of the determinant.

$$\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix} \begin{matrix} a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \end{matrix}$$

Step 2: Find the sum of the products of the elements in each of the three diagonals running down from left to right.

Find $(a_1)(b_2)(c_3) + (b_1)(c_2)(a_3) + (c_1)(a_2)(b_3)$.

Step 3: Find the products of the elements in each of the three diagonals running down from right to left, switch the sign of each product and find the algebraic sum.

Find $-(c_1)(b_2)(a_3) - (a_1)(c_2)(b_3) - (b_1)(a_2)(c_3)$.

Step 4: Add the results found in Steps 2 and 3. That is, find

$(a_1)(b_2)(c_3) + (b_1)(c_2)(a_3) + (c_1)(a_2)(b_3) - (c_1)(b_2)(a_3) - (a_1)(c_2)(b_3) - (b_1)(a_2)(c_3)$

Example

Evaluate
$$\begin{vmatrix} 1 & 2 & -1 \\ 2 & -1 & 3 \\ -3 & -4 & 5 \end{vmatrix}$$

Solution
$$\left|\begin{array}{ccc|cc} 1 & 2 & -1 & 1 & 2 \\ 2 & -1 & 3 & 2 & -1 \\ -3 & -4 & 5 & -3 & -4 \end{array}\right.$$

Step 1: $(1)(-1)(5) + (2)(3)(-3) + (-1)(2)(-4) = -5 - 18 + 8 = -15$

Step 2: $-(-1)(-1)(-3) - (1)(3)(-4) - (2)(2)(5) = 3 + 12 - 20 + -5$

Step 3: $-5 - 15 = \underline{20}$

SYSTEMS2:

Function: to evaluate a system of two equations and two unknowns using Cramer's method.

Input: a system of two equations with two unknown in x and y.

Output: the values of x and y along with the intermediate steps of determinant evaluation.

When determinants are used to solve the system $\begin{cases} a_1x + b_1y = c_1 \\ a_2x + b_2y = c_2 \end{cases}$

the procedure is known as Cramer's method. Thus (102: pp. 274):

$$x = \frac{\begin{vmatrix} c_1 & b_1 \\ c_2 & b_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}}, \qquad y = \frac{\begin{vmatrix} a_1 & c_1 \\ a_2 & c_2 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 \\ a_2 & b_2 \end{vmatrix}}$$

provided the determinant of the denominator is not zero.

Example: Solve $3x + 2y = 10$
$x - 3y = -7$ using determinants

$$x = \frac{\begin{vmatrix} 10 & 2 \\ -7 & -3 \end{vmatrix}}{\begin{vmatrix} 3 & 2 \\ 1 & -3 \end{vmatrix}} \qquad y = \frac{\begin{vmatrix} 3 & 10 \\ 1 & -7 \end{vmatrix}}{\begin{vmatrix} 3 & 2 \\ 1 & -3 \end{vmatrix}}$$

$$x = \frac{(-30) - (-14)}{-11} \qquad y = \frac{(-21) - (10)}{(-9) - 2}$$

$$x = \frac{-16}{-11} \qquad y = \frac{-31}{-11}$$

The solution set is $\left\{ \frac{16}{11} , \frac{31}{11} \right\}$

SYSTEMS3

Function: to solve a system of three equations and three unknowns using Cramer's method.

Input: three equations in three unknowns in x, y and z. Each equation must be on a separate line.

Output: The values of x, y and z along with the intermediate results.

If the three equations are:

$$a_1x + b_1y + c_1z = d_1$$
$$a_2x + b_2y + c_2z = d_2$$
$$a_3x + b_3y + c_3z = d_3$$

then:

$$x = \frac{\begin{vmatrix} d_1 & b_1 & c_1 \\ d_2 & b_2 & c_2 \\ d_3 & b_3 & c_3 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}} \qquad y = \frac{\begin{vmatrix} a_1 & d_1 & c_1 \\ a_2 & d_2 & c_2 \\ a_3 & d_3 & c_3 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}} \qquad z = \frac{\begin{vmatrix} a_1 & b_1 & d_1 \\ a_2 & b_2 & d_2 \\ a_3 & b_3 & d_3 \end{vmatrix}}{\begin{vmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{vmatrix}}$$

provided the denominator is not zero.

Example: Solve by Cramer's method:

$$\begin{cases} x + y + z = 2 \\ 2x - y + 2z = 7 \\ x - 3y - 3z = -2 \end{cases}$$

Solution:

$$x = \frac{\begin{vmatrix} 2 & 1 & 1 \\ 7 & -1 & 2 \\ -2 & -3 & -3 \end{vmatrix}}{\begin{vmatrix} 1 & 1 & 1 \\ 2 & -1 & 2 \\ 1 & -3 & -3 \end{vmatrix}} \qquad y = \frac{\begin{vmatrix} 1 & 2 & 1 \\ 2 & 7 & 2 \\ 1 & -2 & -3 \end{vmatrix}}{\begin{vmatrix} 1 & 1 & 1 \\ 2 & -1 & 2 \\ 1 & -3 & -3 \end{vmatrix}} \qquad z = \frac{\begin{vmatrix} 1 & 1 & 2 \\ 2 & -1 & 7 \\ 1 & -3 & -2 \end{vmatrix}}{\begin{vmatrix} 1 & 1 & 1 \\ 2 & -1 & 2 \\ 1 & -3 & -3 \end{vmatrix}}$$

$$x = \frac{12}{12} \qquad y = \frac{-12}{12} \qquad z = \frac{24}{12}$$

The solution set is (1, -1, 2) .

Tools of algebra was selected in the main module. This module makes possible the selection of four other modules, namely: DETERM2, DETERM3, SYSTEMS2 AND SYSTEMS3. A brief account of each of these modules is given below:

DETERM2:

* displays prompts for the user to enter the elements of the determinant.
* reads these elements into a square 2 x 2 matrix.
* executes the steps for evaluating a 2 x 2 determinant.
* displays the results along with the intermediate steps.
* the process is repeated until either the command "/*" is issued.
* control is returned to the calling module -- Tools of Algebra.

DETERM3:

* displays prompts for the user to enter the elements of the determinant.
* reads these elements into a square, 3 x 3, matrix.
* executes the steps for evaluating a 3 x 3 determinant through the use of various FORMAC73 functions.
* displays the results along with the intermediate steps.
* the process is repeated until "/*" is issued.
* control is returned to the calling program Tools of Algebra.

SYSTEMS2:

* displays the prompts for the user to enter the two equations in x and y.
* the equations are read via another module READER.
* the equations are displayed as an echo-check.
* the coefficients of each equation as well as the right hand side are extracted and then stored into a 2 x3 matrix.
* three 2 x2 matrices are created to represent the numerator determinants for the

calculation of x and y and the denominator determinant.

* the resulting determinants are evaluated with the intermediate results displayed.
* finally, the values of x and y are calculated and displayed.
* this process is repeated until either "/*" is entered.
* control is returned to the calling module Tools of Algebra.

SYSTEMS3:

* displays prompts for the user to enter the three equations in three unknowns: x, y and z.
* the equations are read via READER module.
* the equations just read are displayed as an echo-check.
* the coefficients of the equations and the right hand side are extracted and stored into a 3 x 4 matrix.
* from this matrix, four matrices are created to be used as numerators and a denominator.
* the three numerator matrices are constructed according to Cramer's rules for solving a system of three equations and three unknowns 2 .
* each of the numerator matrices and the denominator matrix are evaluated as determinants.
* the resulting values from the above evaluations are used in calculating the values for x, y and z.
* the intermediate steps generated during the evaluation of the determinants are displayed.
* the final solution is displayed.
* this process is repeated until a "/*" is issued.
* control is returned to the calling module (Tools of Algebra).

ADDITIONAL TOOLS OF ALGEBRA

Function: to allow the selection of the following problem types:

* evaluate permutations using formulas of permulations.
* evaluate combinations using formulas of combinations.
* multiplication, addition, and subtraction of complex numbers.
* division by a complex number.
* reciprocal of a complex number.
* expansion of a binomial expression using the binomial expansion theorem.
* determining the r^{th} term of a binomial expansion.

Input: Any integer in the range of 1 to 7 would constitute a valid entry. Anything outside this boundary will be flagged and the user will have two additional chances to enter a valid selection.

Output: The result of a valid selection will place the user in the desired environment— one that would allow one to practice solving the selected type of problem.

COMBINE

Function: This module will evaluate combinations.

Input: Elements of the combination to be evaluated separated by a comma. For example 52,13 to, mean c(52, 13),

Output: The combination of the given pair.

A combination is a selection of objects considered without regard to their order (12).

The number of n objects taken r at a time will be denoted by c(n, r); where $r \leq n$, r and n are non-negative integers. It is calculated by the following formula:

$$c(n,r) = \frac{n!}{(n-r)!\ r!}$$

Example 1:

Evaluate $$c(4,3) = \frac{4!}{3!\ (4-3)!}$$

$$= \frac{4!}{3!\ 1!} = \frac{4 \cdot 3!}{3!\ 1!}$$

PERMUTE

Function: Evaluate permutations using the formulas of permutation.

Input: the two numbers to be permuted separated by a comma. For example 8,4 to mean p(8, 4).

Output: the result of the permutation.

Permutation is an arrangement of the elements of the set in a specified order (12).

The number of permutations of a set of n different objects taken r at a time, without repetition is:

$$p(n,r) = \frac{n!}{r!}$$

Example

Find the number of permutations of 5 things taken 3 at a time.

Solution:

$$p(5,3) = \frac{5!}{3!}$$

$$= \frac{5 \cdot 4 \cdot 3!}{3!}$$

$$= \underline{20}$$

COMPLEXD

Function: to effect division by a complex expression (or number).

Input: the complex expression to be evaluated.

Output: the result of the evaluation.

When dividing complex expressions (or numbers), one may first determine the product of the denominator and its conjugate, and then divide the numerator by a single real number.

Example: $\dfrac{2}{3 - 2i}$

Solution:

$$\frac{2}{3 - 2i} = \frac{2\,(3 + 2i)}{(3 - 2i)(3 + 2i)}$$

$$= \frac{2\,(3 + 2i)}{3 - 4i^2}$$

$$= \frac{2\,(3 + 2i)}{9 - 4(-1)}$$

$$= \frac{2\,(3 + 2i)}{13}$$

$$= \frac{6 + 4i}{13}$$

$$= \frac{6}{13} + \frac{4}{13}i$$

EVALUATE:

Function: to evaluate a complex expression by performing the necessary operations, namely: complex addition, complex subtraction and multiplication of complex numbers.

Input: complex expression.

Output: the result of the evaluation of the complex expression.

THEORY

The following basic definitions form the basis for evaluating complex numbers:

$\sqrt{-1} = i$

$i^2 = -1$

$i^3 = -i$

$i^4 = 1$

Addition of complex numbers:

If a, b, c, d$\in$ R and a + bi and c + di c, then (a + bi) + (c + di) = (a + c) + (b + d)i.

Example:

$(4 + 2i) + (3 + 3i) = 7 + 5i$

Multiplication of complex numbers:

If a, b, c, d$\in$ R and (a + bi) and c + di$\in$C, then (a + bi) * (c + di) = (ac - bd) + (ad + bc)i.

Recall that $(a + bi) * (c + di) = (ac - bd) + (ad + bc)i$.

If $(a + bi)$ and $(c + di)$ are treated as binomials, then

$$(a + bi) * (c + di) = (a + bi) * c + (a + bi) * di$$

$$= ac + bci + adi + bdi^2;$$

and since $i^2 = -1$

$$= ac + bci + adi - bd$$

$$= (ac - bd) + (bc + ad)i$$

Example: $(3 + 2i)(2 + 4i) = 6 + 12i + 4i + 8i^2$

$$= 6 + 16i + 8(-1)$$

$$= (6 - 8) + 16i$$

$$= \underline{-2 + 16i}$$

BINOMIAL

Function: to expand a binomial expansion using the binomial expansion formula.

Input: a binomial expression.

Output: the expanded binomial expression.

The binomial theorem (102: pp. 306).

If n N,

$$(x + y)^n = x^n + nx^{n-1} y + \frac{n(n-1)}{2} x^{n-2}y^2 + \frac{n(n - 1)(n - 2)}{6} x^{n-3} y^3$$

$$+ \quad + y^n$$

Example 1: Find the expansion of $(2x + b)^3$.

Solution: $(2x + b)^3 = (2x)^3 + 3(2x)^2b^1 + 3(2x)(b)^2 + (b)^3$

$= 8x^3 + 12x^2b + 6xb^2 + b^3$

Example 2: Find the expansion of $(x - 1)^4$.

Solution: $(x - 1)^4 = x^4 + 4(x)^3(-1)^1 + 6(x^2)(-1)^2 + 4(x)^1(-1)^3 + (-1)^4$

$= x^4 - 4x^3 + 6x^2 - 4x + 1$

RECIPROC:

Function: to determine the reciprocal of a complex number or expression.

Input: the complex number or the complex expression.

Output: the reciprocal in a simplified form.

To find the reciprocal of a complex number in standard form we would multiply the numerator and the denominator by the conjugate of the denominator.

Example: Find the reciprocal of $5 + i$

$$\frac{1}{5+i} = \frac{(1)\cdot(5-i)}{(5+i)(5-i)}$$

$$= \frac{5-i}{25+1}$$

$$= \frac{5-i}{26}$$

$$= \frac{5}{26} + \frac{1}{26}i$$

RTHTERM:

Function: to determine the r^{th} term of a given binomial expression.

Input: a binomial expression.

Output: the r^{th} term of the given binomial expression.

THEORY

the r^{th} term of a binomial expansion is:

$$\frac{n(n-1)(n-2)\ \ldots\ (n-r+2)}{(r-1)!}\ x^{n-r+1}\ y^{r-1}$$

Example: Find the fourth term of $(2x^2 - 1)^9$.

$$\frac{}{}\ (2x^2)(-1)$$

$$\frac{9 \cdot 8 \cdot 7 \cdot 6}{4!}\ (2x^2)(-1)$$

$$\frac{9 \cdot 8 \cdot 7 \cdot 6}{4!}\ (2x^2)^5(-1)^4$$

$$\frac{9 \cdot \cancel{8} \cdot 7 \cdot \cancel{6}^2}{\cancel{4} \cdot \cancel{3} \cdot \cancel{2} \cdot 1}\ (2x^2)^5(-1)^4$$

$$= 4032x^{10}$$

Additional tools of algebra module is selected from the menu. This module also aids in the selection of any of the 7 moduels which solve the problems under this area. These modules are shown in Figure 2.2.12 and are as follows: COMBINED, PERMUTE, EVALUATE, COMPLEXD, RECIPROC, BINOMIAL, AND RTHTERM. Each of these are briefly discussed below.

COMBINED

* displays prompts asking the user to enter a problem of this type.
* the problem is read via module READER
* the combination of each pair is evaluated before any other operation is done if there were other operations
* if there are other operations, then they too are evaluated
* each combination evaluated will be displayed
* the result of applying the given operation would be displayed last as it would be the final answer
* this process will be repeated until the "/*" has been issued at which point control is returned to the calling module, additional tools of algebra.

RTHTERM

* displays the prompts for the user to enter a binomial expression
* the expression is read as a character string via another module, READER
* the resulting character string is converted into a FORMAC73 expression
* the formula for finding the r^{th} term is applied, and then the result is displayed
* this whole process is repeated until "/*" is issued.

RECIPROC

* displays the prompts for the user to enter the problem whose reciprocal is to be determined
* the problem is read via another module, READER, as a character string.
* the resulting string is converted into a FORMAC73 expression
* the conjugate of the demoninator is identified and displayed
* the numerator is multiplied by the conjugate and then displayed
* the denominator is multiplied by the conjugate and the result is displayed
* the numerator is divided by the new denominator to yield the final result which is then displayed
* this whole process will be repeated until "/*" is ussued, at which point control is returned to the calling module, additional tools of Algebra.

BINOMIAL

* displays prompts for the user to enter a binomial expression to be expanded
* the expression is read as a character string via the READER module
* the character string is converted into a FORMAC73 expression using PL/1-FORMAC73 interface facilities
* the expression is evaluated using FORMAC73 features
* the final result is displayed
* this whole process is repeated until "/*" is issued, at which point, control is returned to the calling module, additional tools of Algebra

EVALUATE

* displays the prompt for the user to enter a problem of this type
* the prolbem is read via another module, READER
* the string is converted into a FORMAC73 expression
* the expression is evaluated
* the final result is displayed
* this process is continued until "/*" is issued, at which point, control is returned to the calling module, additional tools of Algebra.

COMPLEXD

* displays prompts asking the user to enter a problem of this type
* the problem is read via another module, READER
* the denominator is identified and displayed
* the conjugate of the denominator is determined and displayed
* the numerator is multiplied by the conjugate and the result is displayed
* the denominator is multiplied by the conjugate and the result displayed
* the numerator is then divided by the new denominator (real)
* the final result is displayed
* this process will continue until "/*" is issued, at which point, control is returned to the calling module, additional tools of Algebra.

PERMUTE

* displays prompts asking the user to enter a problem of this type
* the problem is read via another module, READER
* the permutation of each pair is evaluated before any other operation is performed if, indeed, there are other operations
* if there are other operations, namely, multiplication, addition, subtraction, division, and raising to a power, then they are evaluated last
* each permutation evaluated is displayed
* the result obtained after applying all expressions is displayed last
* this process will be repeated until "/*" has been issued, at which point, control is returned to the calling module, additional tools of Algebra.

TRIGONOMETRY

Function: To allow for the selection of the various modules defined to solve trigonometric problems. At the moment, not too many such moudles have been developed. This module, therefore, serves simply as a stub to enhance future expansion. A single module is, however, called by this procedure to expand binomial expressions in trigonometry. For example, $(\cos(x) + \sin(x)) \cdot (\cos(x) - \sin(x)) = \cos^2(x) + \sin^2(x)$.

Input: Will select the only module available.

Output: The user will be placed in the area that allows him/her to perform the desired drill.

DIFFERENTIAL CALCULUS:

Function: to calculate the derivative of a given function using the product rule, the quotient rule, the power rule, and the chain rule. It will also determine higher order derivatives.

Input: A mathematical function entered anywhere on an 80-column record, and separated by a comma from the variable of differentiation. In the case of higher order derivatives, the variable of differentiation will be followed by another comma, and then the integer representing the order of differentiation.

Output: The derivative of the function with respect to the stated variable.

Geometrically, the derivative of a function $y=f(x)$ is a second function that gives the slope of the tangent line to the graph of the first function at each point. In this paper, only the analytic definition will be dicussed.

DEFINITION:

Derivative. If the limit

$$(1) \qquad \lim_{h \to o} \frac{f(x_1+h)-f(x_1)}{h}$$

exists, then that limit is called the derivative of $f(x)$ at $x=x_1$, and is written $f(x_1)$ (read "f prime of x sub-one"). When the limit (1) exists, the function is said to be differentiable of x_1.

Example 1

Find the derivative of $f(x)=3x^2$ at $x=x_1$.

Solution: Applying the definition to the particular function $3x^2$, we have

$$
\begin{aligned}
f'(x) &= \lim_{h \to o} \frac{3(x_1+h)^2-3x_1^2}{h} = \lim_{h \to o} \frac{3x_1^2+6x_1h+3h^2-3x_1^2}{h} \\
&= \lim_{h \to o} \frac{6x_1h+3h^2}{h} = \lim_{h \to o} \frac{\cancel{h}(6x_1+3h)}{\cancel{h}} = 6x_1+3x0 \\
&= 6x_1.
\end{aligned}
$$

Hence $f'(x_1)$, the derivative of this function at x_1, $13x_1$.

The subscript on x serves to remind us that x is fixed at x_1 and that it is h that is varying in the limit process.

The definition of the derivative can be stated in a number of different ways. First, one can condense the definition (1) by writing that

$$(2) \qquad f'(x_1) = \lim_{h \to o} \frac{f(x_1+h)-f(x_1)}{h}$$

whenever the limit on the right side exists. In this form x_1 is fixed and the variable is h.

If, on the other hand, we let x be the variable, we set $x = x_1+h$ so that $h = x - x_1$ and $x \to x_1$ as $h \to 0$. Then (2) becomes

$$f'(x) = \lim_{x \to x_1} \frac{f(x)-f(x_1)}{x - x_1}$$

In re-examining the expression in (1), we find that the function is evaluated at $x_1 + h$ and also at x_1. Thus h is really a change in x, as x goes from x_1 to $x_1 + h$. It is convenient to use the symbol x to denote the change in x, so we set h = x. Further, the numerator, $f(x_1 + h) - f(x_1)$, is just the value of the function when x is $x_1 + h$, minus the value of the function at x_1, and this is just the change in y = f(x), so we are justified in calling it y. Then the expression (1) takes the simple form

(3) $$\lim_{x \to 0} \frac{y}{x}.$$

This form suggests an alternative notation for the derivative which turns out to be very convenient. Indeed, in the limiting process, the Greek letters are replaced by their English equivalents and we have (by definition)

(4) $$\frac{dy}{dx} = \lim_{x \to 0} \frac{y}{x}.$$

This new symbol $\frac{dy}{dx}$ is just another symbol for the derivative and is read "the derivative of y with respect to "x" or "dy over dx".

In computing the derivatives from equation (4), it is convenient to break the procedure into four steps.

(I) In f(x), replace x by $x + \Delta x$. Then by the definition of Δy

(5) $$y + \Delta y = f(x + \Delta x).$$

(II) Subtract $y = f(x)$ from (5), obtaining

(6) $\Delta y = f(x + \Delta x) - f(x).$

(III) Divide both sides of (6) by x, obtaining

(7) $$\frac{\Delta y}{\Delta x} = \frac{f(x + x)-f(x)}{\Delta x}$$

(IV) Take the limit in (7) as x approaches zero,

$$\frac{dy}{dx} = \lim_{x\to 0} \frac{\Delta y}{\Delta x} = \lim_{x\to 0} \frac{f(x+ x)-f(x)}{\Delta x}$$

Example 2

Find the derivative of $y = 5x + 2x - 7$ using the nation of (4).

Solution: Following the four steps just outlined we have

(I) $$\begin{aligned} y + \Delta y &= 5(x+\Delta x)^2 + 2(x+\Delta x) - 7 \\ &= 5x^2 + 10x(\Delta x) + 5(\Delta x)^2 + 2x + 2(\Delta x) -7 \end{aligned}$$

(II) $$\begin{aligned} \Delta y &= f(x +\Delta x) - (x) \\ &= 5x^2 + 10x(\Delta x) + 5(\Delta x)^2 = 2x - 2(\Delta x) -7 -5x^2 -2x -7 \\ &= 10x(\Delta x) + 5(\Delta x)^2 + 2(\Delta x) \end{aligned}$$

(III) $$\frac{\Delta y}{\Delta x} = \frac{10x (\Delta x) + 5(\Delta x)^2 + 2(\Delta x)}{\Delta x}$$

(IV) $$\frac{dy}{dx} = \lim_{x\to 0} \frac{10x (\Delta x) + 5(\Delta x)^2 + 2(\Delta x)}{\Delta x}$$

$$= \lim_{\Delta x\to 0} \frac{x (5(\Delta x) + 10x +2)}{\Delta x}$$

$$\frac{dy}{dx} = \lim_{\Delta x\to 0} 5(0) + 10x + 2$$

$$\frac{dy}{dx} = 10x + 2$$

Computing the derivatives from the definition can be a very tedious process if the given f(x) is complicated. For our purposes, we are going to use differentiation formulas that have been proved.

Theorem 1

The derivative of a constant is zero,

(8) $\frac{d}{dx}c = 0$

Theorem 2

If n is a positive integer and c is a constant, then

(9) $\frac{d}{dx}cx^{n} = cnx^{n-1}$

Example 3:

$y = 5x + 2x - 7$

$\frac{dy}{dx} = 10x + 2$

Theorem 3

The derivative of the sum of two differentiable functions is the sum of the derivatives of the two functions.

(10) $\frac{d}{dx}(u + v) = \frac{du}{dx} + \frac{dv}{dx}$

Example 4

If, $y = 10x^{7} + \pi x^{3}$, then

$\frac{dy}{dx} = 70x^{6} + 3\pi x^{2}$

Theorem 3 can be extended to the sum of any finite number of functions, as stated in Theorem 4.

Theorem 4

The derivative of the sum of any finite number of differentiable functions is the sum of the derivatives of the functions,

(11) $$\frac{d}{dx}(u_1 + u_z + \dots + U_n)$$

$$= \frac{du_1}{dx} + \frac{du_z}{dx} + \dots + \frac{du_n}{dx}.$$

Example 5

If $y = 3x^4 - 8x^3 + 6x^2 - 5x + 18$, find $\frac{dy}{dx}$

$$\frac{dy}{dx} = 12x^3 - 24x^2 + 12x - 5$$

Theorem 5

The derivative of the product of two differentiable functions is the first times the derivative of the second plus the second times the derivative of the first,

(12) $$\frac{d}{dx}uv = u\frac{dv}{dx} + \frac{du}{dx}$$

Example 6

If $y = (x^4 + 3)(3x^3 + 1)$, find $\frac{dy}{dx}$

$$\frac{dy}{dx} = (x^4 + 3)\frac{d}{dx}(3x^3 + 1) + (3x^3 + 1)\frac{d}{dx}(x^4 + 3),$$

$$\frac{dy}{dx} = (x^4 + 3)(9x^2) + (3x^3 + 1)(4 \times 3)$$

$$= 9x^6 + 27x^2 + 12x^6 + 4x^3$$

$$= 18x^6 + 4x^3 + 27x^2$$

Theorem 6

The derivative of the quotient of two differentiable functions is the denominator times the derivative of the numerator minus the numerator times the derivative of the denominator, all divided by the square of the denominator, if the denominator is not zero,

(13) $$\frac{d}{dx}\left(\frac{u}{v}\right) = \frac{\frac{du}{dx} - \frac{dv}{dx}}{v^2}, \quad v = 0$$

Example 7

If, $y = \frac{x}{x^2 - 3x + 5}$, find $\frac{dy}{dx}$,

$$\frac{dy}{dx} = \frac{(x^2 - 3x + 5)\frac{d}{dx}x - x\frac{d}{dx}(x^2 - 3x + 5)}{(x^2 - 3x + 5)^2}$$

$$= \frac{(x^2 - 3x + 5)(1) - x(2x - 3)}{(x^2 - 3x + 5)^2}$$

Theorem 7

The Chain Rule. If $y = f(u)$ and $u = (gx)$ are two differentiable functions then the derivative of the composite function $y = f(g(x))$ is given by (14)

Example 8

Find $\frac{dy}{dx}$ if $y = \frac{x^2 - 1}{x^2 + 1}^5$

Solution: Here we have $y = u^5$, where u is a quotient of two functions of x.

Then,

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx} = 5u^4 \frac{d}{dx}\frac{x^2 - 1}{x^2 + 1}$$

$$= 5\left(\frac{x^2 - 1}{x^2 + 1}\right)^4 \frac{(x^2 + 1)\frac{d}{dx}(x^2 - 1) - (x^2 - 1)\frac{d}{dx}(x^2 + 1)}{(x^2 + 1)^2}$$

$$= 5\left(\frac{x^2 - 1}{x^2 + 1}\right)^4 \frac{(x^2 + 1)(2x) - (x^2 - 1)(2x)}{(x^2 + 1)^2}$$

$$\frac{dy}{dx} = \frac{5(x^2 - 1)^4}{(x^2 + 1)^4} \cdot \frac{(x^2 + 1)(2x) - (x^2 - 1)(2x)}{(x^2 + 1)^2}$$

$$= \frac{10x(x^2 - 1)^4}{(x^2 + 1)^4} \frac{(x^2 + 1)^2 - (x^2 - 1)}{(x^2 + 1)^2}$$

$$\frac{dy}{dx} = 20x\frac{(x^2 - 1)^4}{(x^2 + 1)^4}$$

If we apply the chain rule to y = u we obtain

Theorem 8

If n is any integer and u(x) is any differentiable function, then

$$\frac{d}{dx} u^n = nu^{n-1} \frac{du}{dx} \tag{15}$$

Corollary

If n is any integer, then

$$\frac{d}{dx} (x - a)^n = n(x - a)^{n-1}$$

Example 9

If $y = (3x + 5)^{10}$, find $\frac{dy}{dx}$

$$\frac{dy}{dx} = 10(3x + 5)^a \frac{d}{dx} (3x + 5)$$

$$= 10(3x + 5)^9 (3x)$$

$$\frac{dy}{dx} = 30x (3x + 5)^9$$

Process

In the section that discusses the underlying theory, two methods of computing the derivative are discussed, namely:

(1). using the limit concept.
(2). using the differentiation formulas.

1. Using the limit concept:

In Example 2, a series of specific steps were established. These are the steps that one would perform to get the derivative of a function by using the limit concept. Certainly, the computer can be programmed to follow similar steps in calculating the derivative of a function. This paper, however, will not use this method because:

(1) space limitations would inhibit the long calculations. For example, to find the derivative of $y=(3x+2)^{200}$, the computer will have to expand this function first before completing the first of the four steps that it must go through to obtain the derivative. This is due to the limitations on the internal data structures, will cause a run time error, and the job will abort.

(2) too much CPU time would be spent on unnecessary calculations.

(3) the final result will probably be too cumbersome and long to be useful; and

(4) the form of the finished product will be unnatural.

2. Using differentiation formulas

This paper uses this method exclusively to evaluate derivatives of functions. This method was chosen primarily because it is the most widely used method in analytical differentiations. Using this method, the derivative of $y = (3x+2)^{200}$ is easily computed and reported as $600(3x+2)^{199}$.

DIFFCALC

* Displays prompts for the user to enter the function to be differentiated along with the variable of differentiation.

* Reads the function as a character string via another module, READER.

* Displays the function as the character string after some processing.

* Converts the function into a FORMAC73 expression so that it may be manipulated mathematically.

* Displays the function as a FORMAC73 expression.

* Repeats this whole process starting from the beginning until "/*" is issued, at which point control is returned to the calling module, TUTORIAL.

INTEGRAL CALCULUS:

Function: to calculate the integral of a polynomial with integral exponents. The function to be evaluated must not be a polynomial with variable coefficients. All trigonometric functions, hyperbolic functions, and other mathematical functions must not be a part of the polynomial function to be integrated.

Input: (1) the polynomial to be integrated.

(2) the variable of integration on the same line (record) as the polynomial, but separated by a comma.

Output: The integral of the given polynomial plus a constant of integration.

Integration is the inverse of differentiation, discussed under differential calculus. In other words, given

$$(1) \quad \frac{dy}{dx} = f(x)$$

We are to find a function $y = F(x)$ such that on differentiating $F(x)$ we obtain $f(x)$ (104: pp. 222). If such a function $F(x)$ exists it is called an indefinite integral of $f(x)$. It is also called an antiderivative because it is obtained by reversing the process of differentiation.

To introduce the notation used for this process we first write (1) in the differential form

$$(2) \quad dy = f(x)dx$$

and then prefix an integral sign in front of both sides, obtaining

$$(3) \quad \int dy = \int f(x)dx$$

Here the integral sign indicates that we are to find a function whose differential is the quantity standing after the sign. The function f(x) is called the integrand.

Example 1

If $dy = x^3dx$,

then integrating both sides we would have

$$(4) \quad \int dy = \int x^3dx$$

$$(5) \quad y = \frac{x^4}{4} + c$$

The left side of (5) is y because the differential of y is dy and the right side of (5) is $\frac{x^4}{4} + c$ because the derivative with respect to x of this function is x^3, and so its differential is x^3dx.

The differential of any constant c is zero, and so the integral is not uniquely determined. Because of the indefiniteness of this constant, the integral in (3) is called an indefinite integral.

Theorem 1

If $F^1(x) = f(x)$, then

$$(6) \quad \int f(x)dx = F(x) + c$$

Since integration is the inverse of differentiation, every differentiation formula may be written as an integral formula. The following integration formulas correspond to the differentiation formulas discussed in the previous section:

$$(7) \quad \int dx = x + c$$

$$(8) \quad \int du = u + c$$

$$(9) \quad \int cf(x)dx = c\int f(x)dx$$

$$(10) \quad \int (f(x) + g(x)\ d\ = \int f(x)dx + \int g(x)dx.$$

$$(11) \quad \int x^n dx = \frac{x^{n+1}}{n+1} + c, \text{ if } n \neq -1$$

$$(12) \quad \int u^n du = \frac{u^{n+1}}{n+1} + c, \text{ if } u \neq -1$$

There are many more integration formulas than these, but these are the only ones that are relevant in this paper. Therefore, implementation of a module to perform integration will be based on these rules.

Process

The method of solution implemented is really a brute force method. In evaluating the integral, the integrand is first expanded if it needs to be. In other words, if the polynomial is a binomial or a trinomial expression with integral exponents, then that binomial or trinomial is expanded first using the multiplicative and distributive properties.

Due to the method of solution used, it is not possible to integrate a polynomial such as $y = (x + 2)^{200}$. The module will attempt to evaluate it by first expanding the expression and then performing integration. In the process of expanding this function, internal space limits imposed by the data structures will be exceeded, and the job will abort.

INTEGRAL

* Displays the prompts for the user to enter the function to be integrated along with the variable of integration.

* Expands the polynomial by using both the multinomial, and the distributive laws.

* Displays the expanded polynomial.

* Integrates the polynomial through pattern matching of the powers of x, and then dividing by that power.

* Displays the integral plus a constant of integration.

* Repeats this process from the beginning until "/*" command is issued, at which point control is returned to the calling module, TUTORIAL.

4. Module Implementation

Top Down Implementation

The following steps were followed during the implementation of the entire program:

1. the top module was coded first.

2. no module was coded until the calling module has been coded (and tested).

3. (Partial) program is tested after each module is coded.

4. to allow testing to proceed, module stubs for modules called but not yet coded were used.

A stub is a temporary module required to allow a partially completed program to execute.

C. Program (Module) Coding

While the rest of the code will not be shown until the Appendix, this section will discuss several issues that led to the generation of the entire program code. In particular, the following areas are discussed:

1. Syntactic conventions for module format
2. Pseudo code
3. Structured programming
4. Program style
5. Variable/Data usage

1. Syntactic Conventions for Module Format

The code for each module is clearly delimited in the source program. Each module is started on a new page and then within that module, blank lines are used selectively to delimit the comments from executable program statements. Comments are used to: specify input and output, specify module function, and describe the use of local variables and/or formal parameters. Sample exhibit 1 shows one of the modules.

2. Pseudo code

Each module is preceded by a pseudo code. The pseudo code is included with the module as a part of the identification block. All the pseudo codes that were used (displayed or otherwise) were written in Process Design Language (or PDL). The following can be said about this language:

1. it is somewhere between English and a programming language,
2. it is used to specify program steps at any level,
3. it has flexible syntax that is only meaningful at a high level. The detailed implementation is provided in the code,
4. it is an alternative to flow charts for algorithm design, and
5. it is independent of any specific programming language.

Refer to exhibit 2.3.1 for an example of a typical pseudo code that accompanys some of the modules. It serves as further documentation to some one wishing to change the logic of the module or for someone wishing to understand the logic of the module. It is for this very same reason that PDL was adopted instead of the flow chart symbolism.

3. Structured programming

Structured programming means different things to different people. In this paper, however, the following meaning is adhered to:

(a). it involves the exclusive use of a small number of well-understood and adequate single-entry, single-exit control constructs.

(b). it is not concerned with whether GOTOs are used, but it is concerned with the way inwhich they are used.

(c). it uses the following basic constucts:

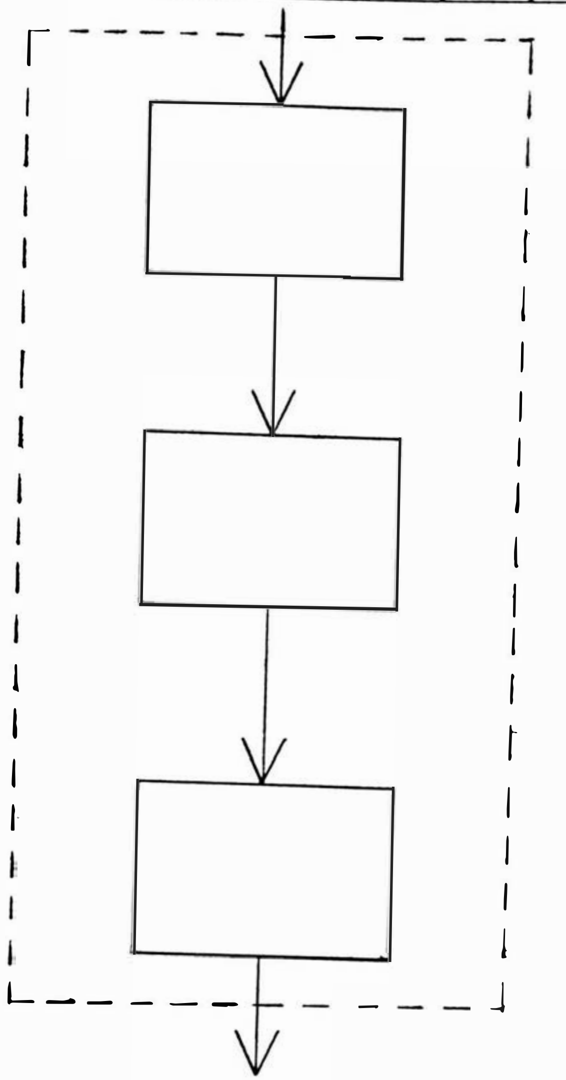
sequence
choice or alternation
repetition or looping

(d). it also uses the following extensions:

*variations on looping
*bounded iteration
*choice with null alternative,
*extended alternation or case

Figures 2.3.3(a) through 2.3.3(e) will serve to clarify each of the control constructs by providing a flow chart as well as a pseudo code for each construct.

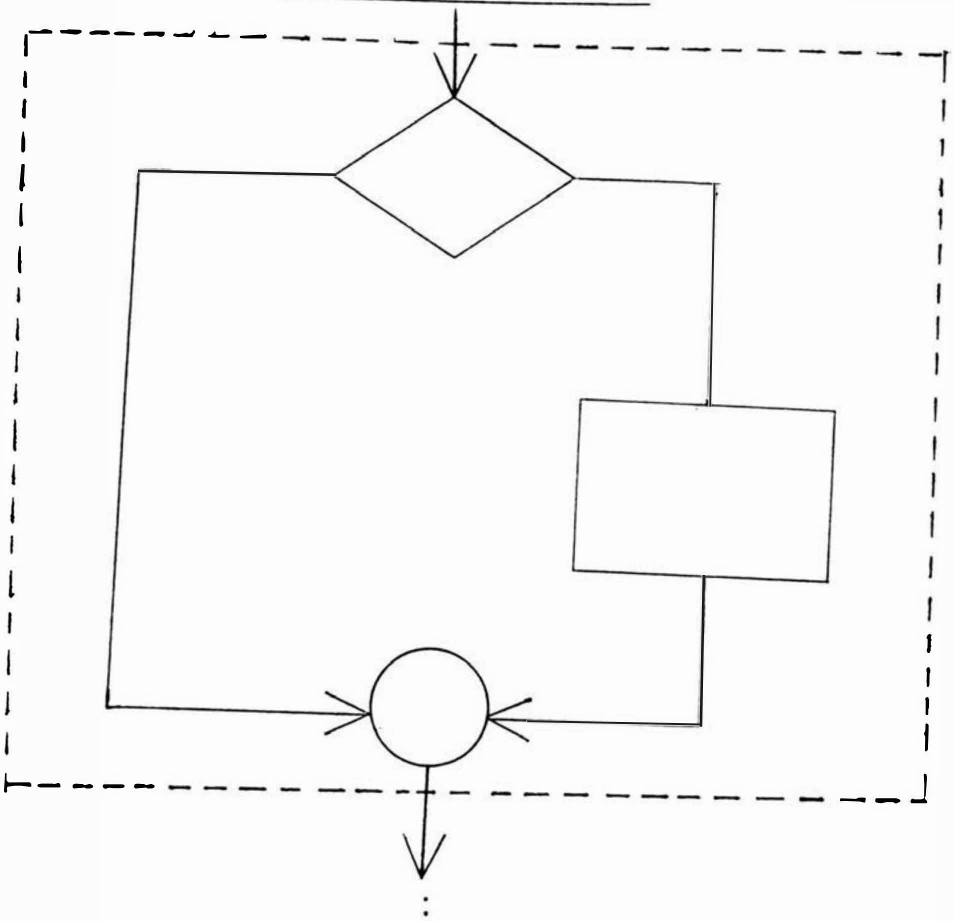
Figure 2.3.2(a) Simple Sequence



Pseudo Code

```
      .
      .
      .
Statement - 1
Statement - 2
Statement - 3

      .
      .
      .
```

2.3.2(b) IF-THEN Statement



IF (condition) THEN

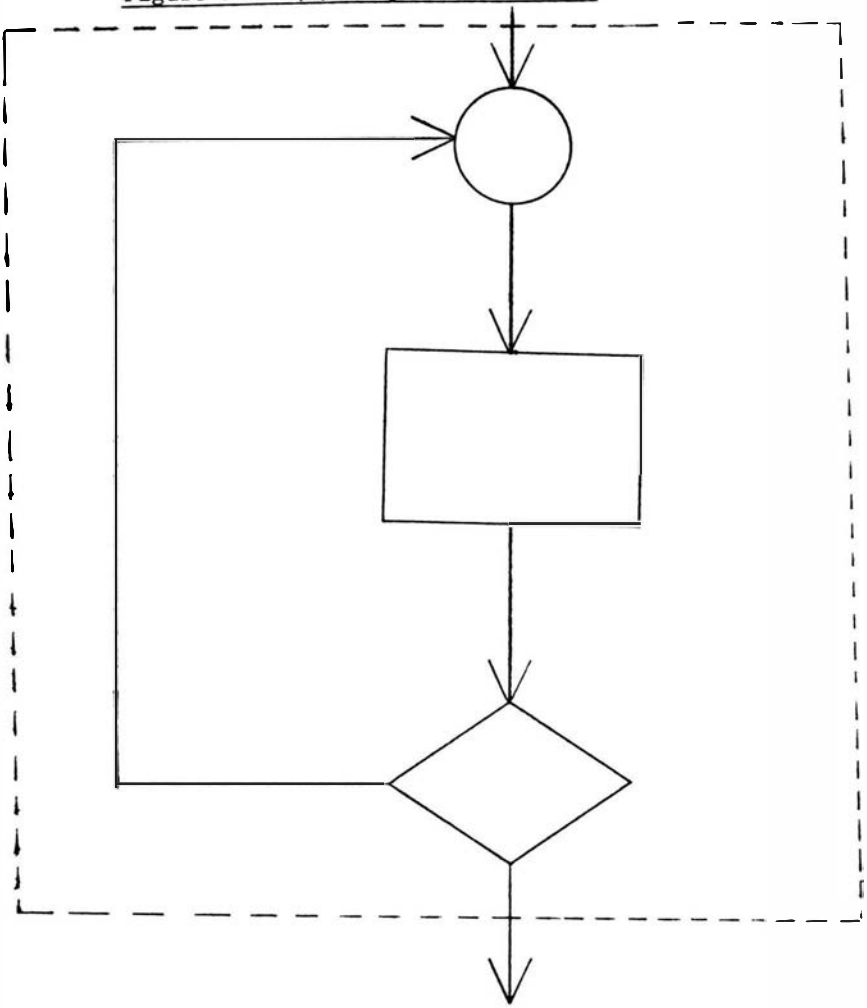
Statement

⋮

Where,

Statement may be a compound statement, i.e. several statements, or another IF - THEN statement, or any other variation, i.e., compound statement, if then and/or if-then-else statements.

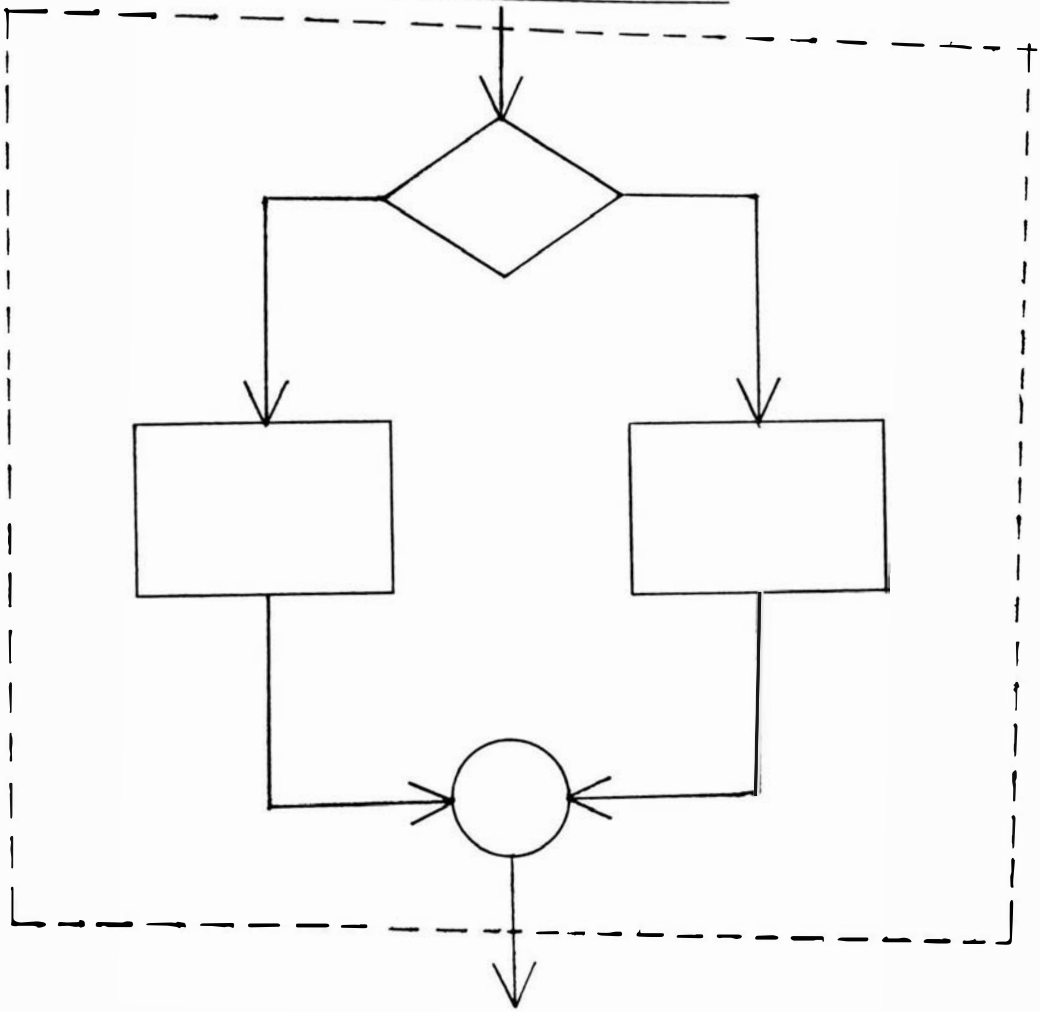
Figure 2.3.2(c) Repeat .... Until



Pseudo Code

```
DO I = Initial Valvue to Final Value
      Statement -1
      Statement -2
           :
      Statement -n
END OF LOOP
```

Figure 2.3.2(d) IF-THEN-ELSE

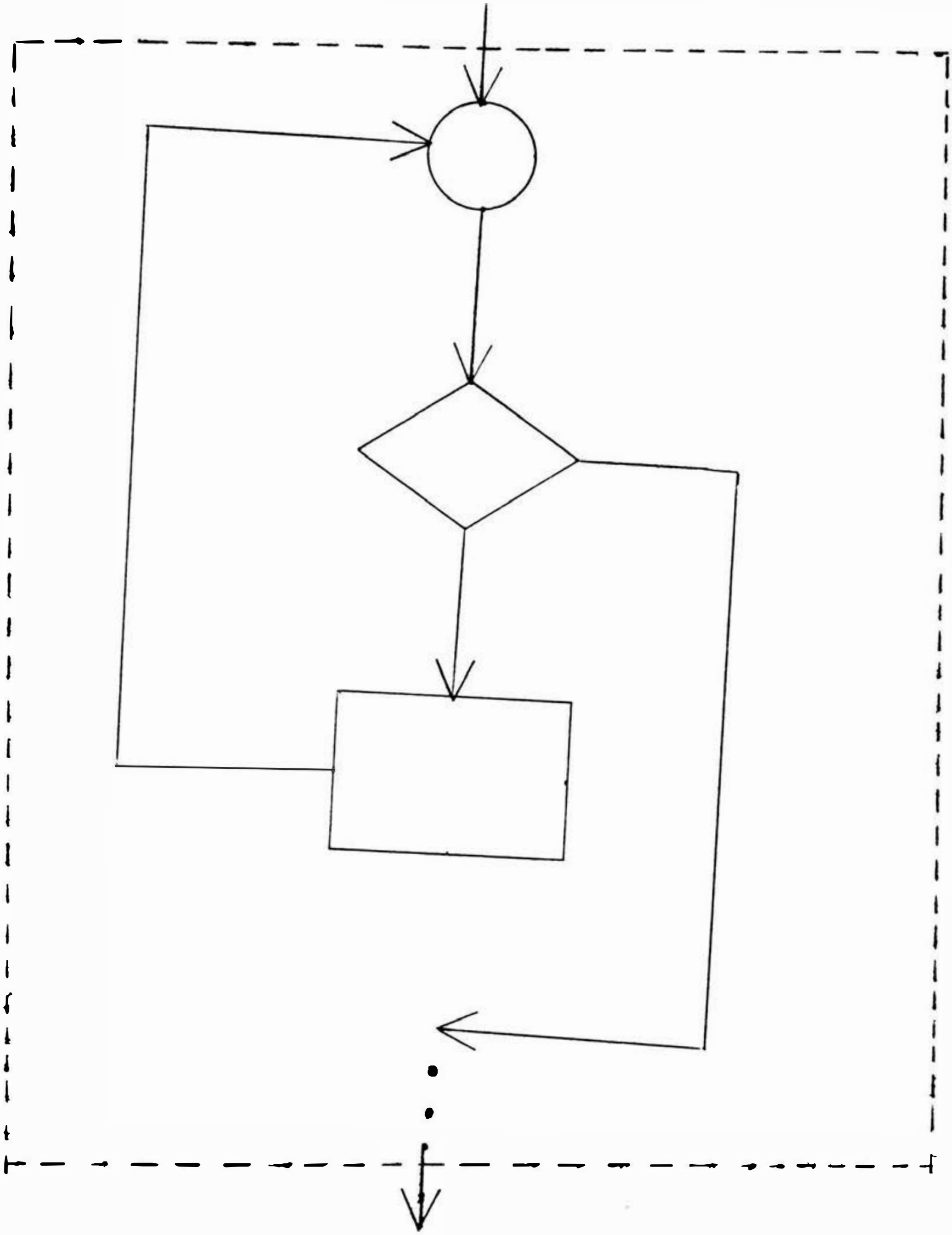


Pseudo Code

```
IF (condition) THEN

Statement -1

ELSE

Statement -2

      .
      .
      .
```

Where Statement -1, and Statement -2 can be compuound statements.

Figure 2.3.2(e) DO-WHILE loop construct



Pseudo Code

```
DO WHILE (condition)

Statement -1
Statement -2
Statement -3
      :
      :
 END OF THE LOOP
```

4. Program Style (Readibility)

Comments, spacing, and indentation are used complementarily to enhance readibility. The following breakdown summarizes how this is achieved.

Comments (in the code)

There are two types of comments: those that are on a separate line (from code), and those that are on the same line as code. The first one serves to tell what is to be done next. It is usually a pseudo code statement starting with a verb or a control keyboard. This should be followed by the programming language statements that implement the pseudo code. The second one should explain what the code does or means in terms of the algorithm.

Spacing (blank lines)

These are used to delimit logical steps.

Identation

The indentation is used to show structure, particularly "THEN" and "ELSE" parts of "IF" statements, and looping constructs. For example:

```
    IF  MORE = TRUE THEN
       CALL DO-MORE (WITH-IT);
    ELSE
       CALL DO-MORE (WITHOUT-IT);
       IF  MORE = TIME THEN
           DO;
       IF (MORE = '1'B) THEN
 THEN-GROUP: DO;
             CALL DO-MORE (WITH-IT);
             PUT SKIP PAGE;
             PUT SKIP EDIT (WITH-IT) (X(5),A);
       END  THEN-GROUP;
    ELSE
ELSE-GROUP: DO;
            CALL DO-MORE (WITHOUT-IT);
            PUT SKIP EDIT (WITHOUT-IT) (X(5),A)
       END  ELSE-GROUP;

LOOP:  DO WHILE (MORE = '1'B);
            GET EDIT (SALES) (F(4))
```

```
    PUT EDIT (SALES) (F(4));
  END LOOP;
```

The code that implements a pseudo code statement comment under the comment should be indented.

Exhibit 2.3.1 shows all these features.

5. Variable/Data Usage

All the variables that are used are declared and listed in comments at the beginning of each module. Global variables are used sparingly, at least not by more than one module. Each variable is used for a single purpose and they are as descriptive as the language constraints permit.

CHAPTER III

Conclusion

A simple tutorial system to perform symbolic mathematics was designed and implemented. The system was modularly designed, and therefore amenable to easy expansion. This feature also makes it possible for the user to be involved with only a subset of the problems at a time. For example, while performing a drill in Algebra, the user may choose a subset of algebra, namely: manipulating fractions. The menu which the package provides was designed for the user's convenience.

Next, it was established through the literature search that there are other symbolic processors. These processors fall in three broad categories. The first category falls under special symbolic processors. These are specific programs designed for specific purposes. Some of these programs include a program to solve problems in quantumelectrodynamics (ASHMEDAI), a program to solve problems in celestial mechanics (TRIGMAN), and many others. In general there are over sixty such programs. The second category falls under general purpose symbolic processors. These are primarily for the main frames due to the substantial memory requirement that they demand. Some of these include REDUCE, MACSYMA, FORMAC73, and SCRATCHPAD. The third category is those symbolic processors that are designed for micro-computers. Some of these include PICOMATH, and muMATH, with PICOMATH being the least powerful of the two. muMATH is powerful enough to solve problems ranging from basic algebra to integral calculus.

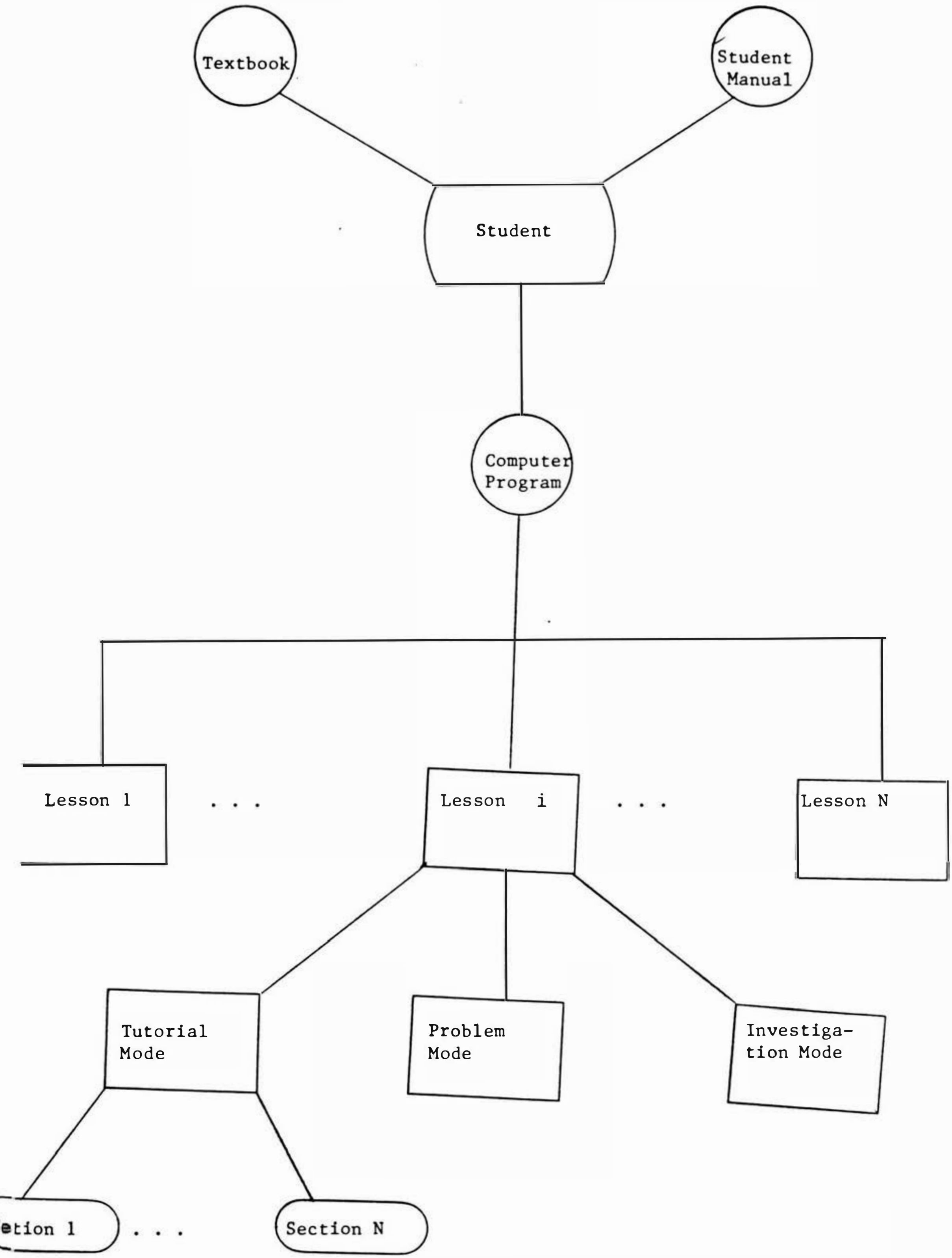
Since the program which the author has developed had the objective of simply investigating the possibility of producing a tutorial drill in symbolic mathematics, there are several things that could be done to enhance its usefulness as a package. For instance, there was no adequate testing by various users to give the author sufficient feedback. Such feedback would be very useful in evaluating the friendliness of the package. It would also help in developing a more extensive error recovery. Very little work was done with trigonometry, primarily because the existing FORMAC73 system of VCU is lacking in some of the modules that would aid in manipulating trigonometric functions. The existing operations in trigonometry are too limited to provide a useful tutorial drill.

It is often necessary to convert results obtained by FORMAC73 into PL/1 before displaying them. This allows for the displaying of descriptive messages about the result before actually printing out the results. While this practice assures that sufficient intermediate steps are displayed, the editing which is usually done by FORMAC73 is lost. This difficulty can be overcome by developing a similar "print out" procedure as the one supplied by FORMAC73, except with an added flexibility. The flexibility would be to add a parameter which will allow for the specification of the result's description, as well as the variable containing the result.

In its current state, the modules are not prepared as lessons. It would be most useful if the package could be re-evaluated and modified accordingly to turn it into a symbolic mathematics package. While there are a variety of ways of doing this, the author recommends the structure shown in Figure 3.1.

The three basic instructional components that would be needed are the text book, the student manual, and the package. The student manual will provide direction on how to use the system, while the text book should be the one being used for classroom instruction. While the package will follow the methods of the text, it will provide the student with the ability to practice as many problems of a given type as necessary. The modules should be designed in lesson form which are consistent with the course objective. Even with the proposed enhancements, this package need not be a static entity. As new developments occur in the world of symbolic mathematics, concurrent with the development of new equipment and the implementation of interfaces with graphic systems, periodic system updates would be necessary.

Figure 3.1 - Structure of the CAI System



APPENDIX A

Appendix A

Computer Aided Instruction

A brief discussion of computer aided instruction (CAI) in given primarily to define some of the terms used in this thesis.

Types, Advantages, Disadvantages and Effectiveness:

Although the literature does not make definitive distinction among CAI packages, they can be classified into two large categories: Adjunct CAI, and Primary CAI. Within Adjunct CAI, it is possible to make further distinctions. Some CAI packages were designed to supplement the learning in the classroom situation, while others substitute for different modes of instruction. Those which supplement the learning situation are referred to as adjunct (8). These are illustrated by short (one-half to one hour) CAI programs, which are used to support or illustrate concepts, and are available through vendor libraries. The concepts illustrated in these programs are then usually discussed in the regular classroom.

In contrast, CAI which provide instruction of a substitute or stand-alone variety are usually of longer duration. These are referred to as the primary CAI. In many discussions worldwide, primary CAI is being debated as a part of distance learning - a term used to describe efforts to provide education to large groups over broad distances or areas. Distance learning often involves many types of educational technology, such as radio, TV, electronic conferencing/mail, and computers. These technologies are used in conjunction with the more traditional methods such as correspondence courses (7).

A second distinction refers to the simplicity-complexity level of CAI. An approach, employing an easy-to-learn programming language as well as minimal hardware to support the use of the programs generated by an author, epitomizes the simplistic approach. PLANIT is an example of such an authoring language. However, such simplistic CAI produces simplistic or limited results; i.e., graphics capabilities, large-scale calculations, and the like are not usually components of such programs. Conversely, complex CAI such as PLATO, which permits extensive use of graphics, large scale calculations, authoring aids, etc., requires complex author languages which necessitate extensive time for authors to adequately learn and, also, requires large-scale computing capability.

Perhaps the most widely accepted value of CAI is that it involves the individual actively in the learning process. It is impossible for the student to be a totally passive member of the situation, and this very activity and involvement facilitates learning (41). Another much touted value is the ability of the learner to proceed at his own pace, which has both implications for the slow learner and the gifted person.

Reinforcement of learning in such situations can be immediate, and systematized, which should result in more effective learning. In addition, the computer, using simulation techniques, permits students to explore time and space, to mix explosive chemicals together in a simulated laboratory without the possibility of destroying themselves and the laboratory, and to investigate complex problems using instruments and methodology which would be excessively costly or not possible at all.

Also, the use of computers in this manner frees faculty members or training coordinators to devote more time to the personal and human considerations of their students (4). Time thus spent with students has been found in a nationwide study of university faculty and students (2) to be the most important factor in students' opinions in the development of their creative abilities. Thus the use of the computer in these modes should result in an educational environment in which individuals learn more and in which their potential for innovative and creative professional work is more fully developed. Similarly, there should be a greater acceptance of the computer as a helpful tool after the student has used simulations, games, or tutorials.

Finally, the problems of handling remedial training for students have increased as evidenced in a report of the National Science Board Commission on Pre-College Education in Mathematics Science, and Technology (19):

> Remedial mathematics enrollments at four-year institutions of higher education increased 72 percent between 1975 and 1980, while a total student enrollments increased by only seven percent. At public four-year colleges, 25 percent of the mathematics courses are remedial; and at community colleges, 42 percent are.

This increase is due to the recognition of the problems of bilingual and disadvantaged students and the inadequate English and mathematics skills of entering university students. Computer tutorials, especially in these areas, appear to be both educationally sound and reasonable in cost, if approached in an appropriate manner. Similar cases can be made for the use of CAI to support continuing education and industrial training programs. Cobot and Holt in their case study of a formative phase of a contract which, in whole, required the design and development of a 256-hour computer based curriculum in Electronics principles concluded that the benefit of the evaluation efforts of the phase studied, however, was that the facts were learned at little cost in either time, money, or ultimately data quality of the project (3).

The disadvantages of using CAI in the learning process can be divided into three main categories. These are:

(1) The need for teachers and training directors to move from accepted methods with which they are familiar to relatively new, untried methods in which most individuals have little expertise and which arouses considerable fear and antipathy owing to its technological base.

(2) The primitive state of the art in CAI, in which a diversity of computing hardware together with various author CAI languages compete with little apparent co-ordination from professionals in the education world. The author's experience in mathematical CAI is that the available CAI course materials are typically poorly constructed, largely undocumented, and able to run on selected computers for which they were written. Also, there are relatively few "experts" to whom beginning CAI users can turn for assistance; and

(3) The cost of hardware, CAI course materials (courseware), and individuals to help implement the process - especially since computer vendors initially touted CAI as an ultimate cost saving device. When used as a substitute or replacement method for learning, CAI can be cost saving; however, in actuality CAI is used today mainly as supplement to enrich learning in the educational scene; and therefore, cost should be considered as add-ons.

The effectiveness of CAI has been defined variously by different investigators. To some effectiveness means the amount of learning that takes place initially. To others, it means the degree of retention of learning, or at the very least, whether or not an individual stays in or drops out of a learning experience. Still others are concerned with the learner's change in attitude toward the computer as a helpful tool in the culture. Finally, owing to the fact that CAI is in its infancy, some are simply concerned with transportability of materials and/or acceptance of materials for use by others.

In general, well-designed, tightly controlled, evaluative studies of the use of CAI are rare. Some have been conducted however; and trends are becoming discernible. Several of the more prominent studies will be reviewed.

The CAI physics course developed at the Florida State University is in the form of a computer tutorial. Tentative evaluations indicate that instructional time was reduced by 17 percent over the traditional lecture course, and students scored higher on final exams and attained superior conceptual mastery (10).

The medical school of the University of Southern California has used computer-controlled modeling to teach anesthesiology. Evaluations using experimental and control groups demonstated that when the model was used, fewer trials over a shorter period of time were required for students to reach an acceptable level of professional performance (10).

Studies of the CAI Russian course at Stanford, using experimental and control groups, revealed positive results in terms of student performance on examinations, student behavior, and student responses to a questionaire about the program. Students taking the computer-based course scored "significantly better" on the final exams. In addition, far fewer students dropped out of the computer-based course (10).

Probably the most significant uses of the computer in simulation, game or tutorial modes are represented by the Chicago City Schools Project (using Suppes and Atkinson's materials), the PLATO project, and the Time Shared Interactive Computer Controlled Information Television (TICCIT) project.

The Chicago City Schools project was began in 1971 and is continuing today. It affects over 12,000 fourth through eighth grade children in the inner city schools, with 850 terminals providing tutorial lessons in mathematics and reading. Originally designed to improve skills in these areas, the project has had significant results. As an example, the average increase in reading ability in schools was 5.4 months per pupil for each 10 months of regular classroom instruction. Using the computer tutorial approach, the average rose to 9.0 months improvement for 8 months of instruction (13). This program is now being formally evaluated by the Educational Testing Service under a grant from the National Institute of Education (13).

SAMPLE SESSION FOR INTRFACE, AND MATHPGM

```
RUN 'MATHPGM'
ENTER THE OUTPUT DEVICE:-
          1.  ---  FOR CONSOLE(NO HARD COPY)
          2.  ---  FOR PR1(FASTER PRINTER)
          3.  ---  FOR PR(DEC-WRITTER
1
HAVE YOU ENTERED SEQUENCE AND FUNCTION DEFINITIONS (YES OR NO)
NO
DEFINE FUNCTIONS ON LINES 500 THROUGH 600
INSERT START DEFINITIONS ON LINES 2000 THROUGH 2100
INSERT ADVANCE DEFINITIONS ON LINES 3000 THROUGH 3100
CALL ME WHEN YOU'RE READY
  STOP 430
500 DEF FNF(X) = 7*X^3 - 4*X^2 - 5*X + 1
2000 LET X0 = A
2010 LET Y0 = FNF(A)
2020 LET D = (B-A)/M
3000 LET X = A + N*D
3010 LET Y = FNF(X)
RUN
ENTER THE OUTPUT DEVICE:-
          1.  ---  FOR CONSOLE(NO HARD COPY)
          2.  ---  FOR PR1(FASTER PRINTER)
          3.  ---  FOR PR(DEC-WRITTER
1
HAVE YOU ENTERED SEQUENCE AND FUNCTION DEFINITIONS (YES OR NO)
YES
ENTER THE NAMES OF SEQUENCES TO BE PRINTED.
SELECT FROM T,U,V,W,X,Y,Z, OR ENTER * FOR BLANK SEQUENCE.
COLUMN 1
X
COLUMN 2
Y
COLUMN 3
Z
COLUMN 4
W
ENTER M, THE NUMBER OF SEQUENCE STEPS
-2     2     0     0
Your input (-2) was interpreted as -2
This input must be greater than or equal to  0
                 and less than or equal to  1000
Please try again.
20
ENTER PARAMETERS A,B,C,D
-2
I was expecting  4  items but I found only  1 .  Remember numbers MUST be s
ted
by blanks and/or a comma.
Please try again.
-2    2    0    0
NUMBER OF STEPS M= 20
A=-2  B= 2  C= 0  D= .2
N              X              Y              Z              W
 0            -2             -61             0              0
 1            -1.8           -43.784         0              0
 2            -1.6           -29.912         0              0
 3            -1.4           -19.048         0              0 (page 119.2)
 4            -1.2           -10.856         0              0
```

```
                         1.784        0          0
          0              1            0          0
          .2            -.104         0          0
          .4            -1.192        0          0
          .6            -1.928        0          0
          .8            -1.976        0          0
          1             -1            0          0
          1.2            1.336        0          0
          1.4            5.368        0          0
          1.6            11.432       0          0
          1.8            19.864       0          0
          2              31           0          0
ou want another run with present definitions

CII

X    -END OF TASK CODE=    0     CPUTIME=14.296/0.666
```

```
500  DEF FNF(X) = 7*X^3 - 4*X^2 - 5*X + 1
2000 LET X0 = A
2010 LET Y0 = FNF(A)
2020 LET D = (B-A)/M
3000 LET X = A + N*D
3010 LET Y = FNF(X)
RUN
ENTER THE OUTPUT DEVICE:-
          1.  ---  FOR CONSOLE(NO HARD COPY)
          2.  ---  FOR PR1(FASTER PRINTER)
          3.  ---  FOR PR(DEC-WRITTER
1
HAVE YOU ENTERED SEQUENCE AND FUNCTION DEFINITIONS (YES OR NO)
YES
ENTER THE NAMES OF SEQUENCES TO BE PRINTED.
SELECT FROM T,U,V,W,X,Y,Z, OR ENTER * FOR BLANK SEQUENCE.
COLUMN 1
X
COLUMN 2
Y
COLUMN 3
Z
COLUMN 4
W
ENTER M, THE NUMBER OF SEQUENCE STEPS
20
ENTER PARAMETERS A,B,C,D
-2    2    0    0
NUMBER OF STEPS M= 20
A=-2  B= 2  C= 0  D= .2
N            X              Y              Z            W
 0          -2             -61             0            0
 1          -1.8           -43.784         0            0
 2          -1.6           -29.912         0            0
 3          -1.4           -19.048         0            0
 4          -1.2           -10.856         0            0
 5          -1             -5              0            0
 6          -.8            -1.144          0            0
 7          -.6             1.048          0            0
 8          -.4             1.912          0            0
 9          -.2             1.784          0            0
 10          0              1              0            0
 11          .2            -.104           0            0
 12          .4            -1.192          0            0
 13          .6            -1.928          0            0
 14          .8            -1.976          0            0
 15          1             -1              0            0
 16          1.2            1.336          0            0
 17          1.4            5.368          0            0
 18          1.6            11.432         0            0
 19          1.8            19.864         0            0
 20          2              31             0            0
Do you want another run with present definitions
NO
BASICII
QUIT
PAL EX   -END OF TASK CODE=   0    CPUTIME=11.312/0.657
LOG
```

```
RUN 'INTRFACE'
                    MENU
ENTER  0.  --- TO SIMPLY SKIM THROUGH THE MENU
ENTER:-
      35.  ---  TO GRAPH ANY FUNCTION
       1.  ---  TO EXECUTE FOX AND RABBIT ECOLOGY MODEL
       2.  ---  TO PRODUCE A SINE TABLE
                SQUARES
       3.  ---  TO OBTAIN SUMS OF RECIPROCALS AND THEIR
                SQUARES
       4.  ---  TO OBTAIN A FIBONACCI SEQUENCE
       5.  ---  TO OBTAIN SQUARE-ROOT BY BABYLONIAN METHOD
1
ENTER THE OUTPUT DEVICE:-
     1.  ---  FOR THE CONSOLE(NO HARD COPY)
     2.  ---  FOR THE LINE PRINTER(PR1:)
     3.  ---  FOR THE DEC-WRITTER(PR:)
1
FOX AND RABBIT ECOLOGY MODEL
LET'S GO WITH THE INITIAL NUMBERS OF RABBITS
AND FOXES. (ENTER X,Y LESS THAN 10.0.)
3      0.5
NOW THE POPULATION GROWTH AND DEATH FACTORS.
(ENTER A,B,C,P, ALSO LESS THAN 10.0)
4    2    1    3
TIME RABBITS FOXES
0.25   611    69
                       F                           R
0.50   904   226
                               F                                 R
0.75   310   551
                                    R           F
1.00    69   382
                       R               F
1.25    44   203
                      R       F
1.50    55   107
                       R F
1.75    99    60
                       F R
2.00   208    40
                      F       R
2.25   458    41
                      F                    R
2.50   914   101
                         F                                         R
2.75   719   496
                                             F          R
3.00    93   533
                         R                     F
3.25    32   285
                      R           F
3.50    30   143
                      R    F
3.75    49    73
                      R F
4.00    99    41
                      F  R
4.25   223    28
                     F         R
```

```
4.75   1082    97
                        F
5.00    659   631
                                                   FR
END OF RUN NUMBER  1
DO YOU WISH TO CHANGE YOUR INPUT
VARIABLES AND RUN AGAIN?
(ANSWER 1 FOR YES, 0 FOR NO)
1
LET'S GO WITH THE INITIAL NUMBERS OF RABBITS
AND FOXES. (ENTER X,Y LESS THAN 10.0.)
3      1
NOW THE POPULATION GROWTH AND DEATH FACTORS.
(ENTER A,B,C,P, ALSO LESS THAN 10.0)
4      2      1      3
TIME RABBITS FOXES
0.25    477   122
                         F                 R
0.50    578   223
                              F                 R
0.75    355   359
                                     F
1.00    164   315
                           R       F
1.25    120   208
                         R   F
1.50    140   133
                          F
1.75    215    96
                        F     R
2.00    368    91
                        F            R
2.25    581   139
                          F                     R
2.50    566   299
                                  F            R
2.75    243   391
                               R       F
3.00    117   279
                         R       F
3.25    103   171
                        R   F
3.50    141   108
                        F R
3.75    241    80
                       F       R
4.00    437    85
                       F                 R
4.25    678   160
                           F                         R
4.50    507   379
                                      F     R
4.75    167   396
                           R           F
5.00     88   249
                       R       F
END OF RUN NUMBER  2
DO YOU WISH TO CHANGE YOUR INPUT
VARIABLES AND RUN AGAIN?
(ANSWER 1 FOR YES, 0 FOR NO)
1
LET'S GO WITH THE INITIAL NUMBERS OF RABBITS
AND FOXES.
3      1.3
```

(Page 119.6)

```
 0.25    413    148
                          F             R
 0.50    466    215
                              F           R
 0.75    352    292
                                  F  R
 1.00    218    279
                              R  F
 1.25    172    211
                            R F
 1.50    188    154
                           FR
 1.75    255    125
                         F      R
 2.00    374    127
                         F            R
 2.25    489    178
                            F              R
 2.50    432    279
                                 F       R
 2.75    254    311
                                R  F
 3.00    166    244
                           R   F
 3.25    160    171
                            F
 3.50    209    126
                         F   R
 3.75    315    112
                         F         R
 4.00    467    139
                          F              R
 4.25    521    234
                               F            R
 4.50    330    334
                                    F
 4.75    177    291
                            R     F
 5.00    139    200
                          R  F
END OF RUN NUMBER  3
DO YOU WISH TO CHANGE YOUR INPUT
VARIABLES AND RUN AGAIN?
(ANSWER 1 FOR YES, 0 FOR NO)
LOG
```

```
RUN "INTRFACE"
                         MENU
ENTER  0.   --- TO SIMPLY SKIM THROUGH THE MENU
ENTER:-
       35.  ---  TO GRAPH ANY FUNCTION
        1.  ---  TO EXECUTE FOX AND RABBIT ECOLOGY MODEL
        2.  ---  TO PRODUCE A SINE TABLE
                 SQUARES
        3.  ---  TO OBTAIN SUMS OF RECIPROCALS AND THEIR
                 SQUARES
        4.  ---  TO OBTAIN A FIBONACCI SEQUENCE
        5.  ---  TO OBTAIN SQUARE-ROOT BY BABYLONIAN METHOD
0
ENTER:-
        6.  ---  TO EXECUTE HARDY'S TAXI PROGRAM
        7.  ---  TO OBTAIN SEQUENCES OF POWERS
        8.  ---  TO OBTAIN SUMS OF GEOMETRIC PROGRESSIONS
       9.  ---  TO OBTAIN POWERS VS. FACTORIALS
       10.  ---  TO OBTAIN DERIVATIVES VIA SEQUENCES
       11.  ---  TO RUN PROGRAM 10.2 FROM THE TEXT
       12.  ---  TO COMPUTE PI BY THE METHOD OF PERIMETERS
       13.  ---  TO FIND ROOTS BY INTERVAL-HALVING METHOD
       14.  ---  TO OBTAIN ROOTS BY NEWTON'S METHODS
       15.  ---  TO OBTAIN THE MAXIMUM OF A FUNCTION BY
                 INTERVAL-HALVING METHOD.
       16.  ---  TO EVALUATE INTEGRALS OF MONOTONE FUNCTIONS
       17.  ---  TO DO INTEGRATION BY TRAPEZOIDAL AND
                MID-POINT RULES
       18.  ---  TO EVALUATE INTEGRALS BY SIMPSON'S RULE
       19.  ---  TO TABULATE FUNCTIONS AND DERIVATIVES
       21.  ---  TO TABULATE FUNCTIONS AND DERIVATIVES
                 WITH AN END TEST TOLERANCE
2
ENTER THE OUTPUT DEVICE:-
      1.  ---  FOR THE CONSOLE(NO HARD COPY)
      2.  ---  FOR THE LINE PRINTER(PR1:)
      3.  ---  FOR THE DEC-WRITTER(PR:)
1
 0              0
 .25            .24740396
 .5             .47942554
 .75            .68163876
 1              .84147098
 1.25           .94898462
 1.5            .99749499
 1.75           .98398595
 2              .90929743
 2.25           .7780732
 2.5            .59847214
 2.75           .38166099
 3              .14112001
 3.25          -.10819513
 3.5           -.35078323
 3.75          -.57156132
 4             -.7568025
 4.25          -.89498936
 4.5           -.97753012
 4.75          -.99929279
 5             -.95892427
 5.25          -.85893449
 5.5           -.70554033
```

(Page 119.8)

```
RUN 'INTRFACE'
                        MENU
ENTER  0.   --- TO SIMPLY SKIM THROUGH THE MENU
ENTER:-
      35.   ---  TO GRAPH ANY FUNCTION
      1.   ---  TO EXECUTE FOX AND RABBIT ECOLOGY MODEL
      2.   ---  TO PRODUCE A SINE TABLE
                SQUARES
      3.   ---  TO OBTAIN SUMS OF RECIPROCALS AND THEIR
                SQUARES
      4.   ---  TO OBTAIN A FIBONACCI SEQUENCE
      5.   ---  TO OBTAIN SQUARE-ROOT BY BABYLONIAN METHOD
5
ENTER THE OUTPUT DEVICE:-
     1.   ---  FOR THE CONSOLE(NO HARD COPY)
     2.   ---  FOR THE LINE PRINTER(PR1:)
     3.   ---  FOR THE DEC-WRITER(PR:)
1
PROGRAM 4.4 (BASIC)
BABYLONION SQUARE-ROOT METHOD.
INPUT A,X
100       2
 26
 14.923077
 10.812054
 10.030495
 10.000046
 10
BASICII
QUIT
PALEX   -END OF TASK CODE=    0     CPUTIME=1.030/0.240
LOG
```

```
RUN "INTRFACE"
                        MENU
ENTER  0.  --- TO SIMPLY SKIM THROUGH THE MENU
ENTER:-
       35.  ---  TO GRAPH ANY FUNCTION
       1.   ---  TO EXECUTE FOX AND RABBIT ECOLOGY MODEL
       2.   ---  TO PRODUCE A SINE TABLE
                 SQUARES
       3.   ---  TO OBTAIN SUMS OF RECIPROCALS AND THEIR
                 SQUARES
       4.   ---  TO OBTAIN A FIBONACCI SEQUENCE
       5.   ---  TO OBTAIN SQUARE-ROOT BY BABYLONIAN METHOD
0
ENTER:-
       6.   ---  TO EXECUTE HARDY'S TAXI PROGRAM
       7.   ---  TO OBTAIN SEQUENCES OF POWERS
       8.   ---  TO OBTAIN SUMS OF GEOMETRIC PROGRESSIONS
      9.   ---  TO OBTAIN POWERS VS. FACTORIALS
      10.   ---  TO OBTAIN DERIVATIVES VIA SEQUENCES
      11.   ---  TO RUN PROGRAM 10.2 FROM THE TEXT
      12.   ---  TO COMPUTE PI BY THE METHOD OF PERIMETERS
      13.   ---  TO FIND ROOTS BY INTERVAL-HALVING METHOD
      14.   ---  TO OBTAIN ROOTS BY NEWTON'S METHODS
      15.   ---  TO OBTAIN THE MAXIMUM OF A FUNCTION BY
                 INTERVAL-HALVING METHOD.
      16.   ---  TO EVALUATE INTEGRALS OF MONOTONE FUNCTIONS
      17.   ---  TO DO INTEGRATION BY TRAPEZOIDAL AND
                MID-POINT RULES
      18.   ---  TO EVALUATE INTEGRALS BY SIMPSON'S RULE
      19.   ---  TO TABULATE FUNCTIONS AND DERIVATIVES
      21.   ---  TO TABULATE FUNCTIONS AND DERIVATIVES
                 WITH AN END TEST TOLERANCE
21
ENTER THE OUTPUT DEVICE:-
      1.   ---  FOR THE CONSOLE(NO HARD COPY)
      2.   ---  FOR THE LINE PRINTER(PR1:)
      3.   ---  FOR THE DEC-WRITER(PR:)

PROGRAM 37.1 (BASIC)
                COMPUTATION OF SIN(X) TO WITHIN E
INPUT X,E
0      2
 X                 SIN X            DEGREE
 0                 0                1
BASICII
QUIT
PALEX    -END OF TASK CODE=    0     CPUTIME=1.087/0.265
LOG
```

```
RE "INTRFACE"
                  MENU
ENTER  0.  --- TO SIMPLY SKIM THROUGH THE MENU
ENTER:-
     35.  ---  TO GRAPH ANY FUNCTION
     1.  ---  TO EXECUTE FOX AND RABBIT ECOLOGY MODEL
     2.  ---  TO PRODUCE A SINE TABLE
              SQUARES
     3.  ---  TO OBTAIN SUMS OF RECIPROCALS AND THEIR
              SQUARES
     4.  ---  TO OBTAIN A FIBONACCI SEQUENCE
     5.  ---  TO OBTAIN SQUARE-ROOT BY BABYLONIAN METHOD
1
ENTER THE OUTPUT DEVICE:-
    1.  ---  FOR THE CONSOLE(NO HARD COPY)
    2.  ---  FOR THE LINE PRINTER(PR1:)
    3.  ---  FOR THE DEC-WRITTER(PR:)
1
FO AND RABBIT ECOLOGY MODEL
LE'S GO WITH THE INITIAL NUMBERS OF RABBITS
AN FOXES. (ENTER X,Y LESS THAN 10.0.)
3   0.5
NO THE POPULATION GROWTH AND DEATH FACTORS.
(ETER A,B,C,P, ALSO LESS THAN 10.0)
4   2    1    3
TIE RABBITS FOXES
0.5   611    69
                    F                           R
0.0   904   226
                            F                                 R
0.5   310   551
                                R           F
1.0    69   382
                    R               F
1.5    44   203
                   R       F
1.0    55   107
                    R F
1.5    99    60
                    F R
2.0   208    40
                   F       R .
2.5   458    41
                   F                   R
2.0   914   101
                     F                                         R
2.5   719   496
                                         F          R
3.0    93   533
                     R                     F
3.5    32   285
                   R         F
3.0    30   143
                   R     F
3.5    49    73
                   R F
4.0    99    41
                   F  R
4.5   223    28
                  F         R
4.    519    71
```

(Page 119.11)

```
4.0   318    31
                  F                               R
4.5  1082    97
                     F
5.0   659   631
                                                        FR

END OF RUN NUMBER  1
DO YOU WISH TO CHANGE YOUR INPUT
VARIABLES AND RUN AGAIN?
(ANSWER 1 FOR YES, 0 FOR NO)
YES
44 ERR 750
1
45 ERR
LO
```

APPENDIX B

APPENDIX B

As stated early on, the combination of the PL/I language and FORMAC73 is used in this thesis mainly due to both being available on the computer system at Virginia Commonwealth University. However, a brief discussion of other symbolic processors is given in this appendix for reference and completeness.

Discussion of Selected Symbolic Processors

Unlike most programs, in which all variables must be set to numbers, Symbolic Mathematics Systems can process variables that are defined as expressions. For example, suppose that the user is trying to find the roots to the quadratic equation $ax^2 + bx + c = 0$ via the quadratic formula:

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

Solving for x with a numeric processor, like FORTRAN, when the program is run, a, b, and c would have to be set equal to numbers. With a symbolic processor, such as MACSYMA, a, b, and c may be set to equal other expressions. For instance, a might equal q, b might equal -pq-1, and c might equal p. The symbolic processor can then substitute these expressions into the quadratic formula:

$$\frac{(pq + 1) + \sqrt{p^2q^2 + 2pq + 1 - 4qp}}{2q}$$

and so on, eventually arriving at the solution:

$$x = p$$
$$x = 1/q$$

In this appendix, five selected symbolic processors are briefly discussed. The discussions of these systems are quite general, but elaborate enough to give a reasonable overview of their functions, and limitations for performing symbolic mathematics. These symbolic mathematics systems are as follows:

(1) muMATH

(2) REDUCE

(3) MACSYMA

(4) SCRATCHPAD

muMATH

The latest release of muMATH that the author is aware of is muMATH-80. As a symbolic mathematics system, muMATH allows the user to explore the world of symbolic mathematices from elementary arithmetic through calculus, and matrix algebra. It is distributed by the SOFT WAREHOUSE in Honolulu, Hawaii. It is run on microcomputers (personal computers). The ability to simplify both numeric and algebraic expressions is a natural extension of the strictly numerical capabilities normally provided by computers and pocket calculators.

Roundoff errors are eliminated by muMATH by using exact rational arithmetic, and makes it possible to get results up to 611 digits of accuracy (20). This high precision would prove invaluable to those who are interested in number theory and data encryption.

muMATH was modularly designed and prepared, and so it can be distributed as a modular set of packages, each providing a set of mathematical capabilities. In much the same way that mathematics is commonly taught, the higher level muMATH packages require more basic packages as a prerequisiste. For example, the logarithm and trigonometry packages provide muMATH with the ability to apply a comprehensive set of transformations, making it possible to simplify expressions containing logarithms and/or trigonometric functions. This can be used to solve trigonometric problems involving logarithms which would otherwise have to be looked up in a table. Next, the equation package provides the ability to solve linear and non-linear equations. When matrix, and array packages are loaded, systems of linear equations can be solved, and matrix inverses, dot products, and determinants can be computed. The matrices may have numeric as well as symbolic elements.

The muMATH calculus package allows one to compute derivatives, integrals, and limits of expressions. Packages for computing the Taylor series of expressions and for determining the closed form summations and product of series are also included.

muMATH is written entirely in the muSIMP-80 programming language, which is the latest release of muSIMP as of this writing. From here on, muSIMP will be used to mean muSIMP80. Therefore, extensions of muMATH must be made using muSIMP. muSIMP is an extensible language suited for symbolic and for semi-numerical processing. It gives the programmer the sematic power of LISP, but with a more familiar syntax, similar to such block-structured languages as PASCAL or PL/1. muMATH is not without limitations. Some of the more basic limitations are briefly mentioned and discussed under the appropriate categories in the interest of simplicity and clarity.

Arithmetic and Algebra

For those problems that muMATH is unable to solve, it responds with the word FALSE. For instance, muMATH cannot solve 300! The author reasons that this is not a muMATH limitations, but rather one that is imposed by the system. The reasoning is based on Stantemyer's statement that rational arithmetic is usually indefinite precision, where each number occupies as much memory as is necessary for exact representation up to some very large maximum, imposed only by the total amount of remaining space allocated for numbers (17).

Next, the package does not exhibit artificial intelligence (although with some of its accomplishments, it seems to exhibit it). It is a very sophisticated symbol manipulator that rigorously applies a given set of rules to arrive at a transformed result. It must be noted that achieving a desired algebraic manipulation is not always an exact process. For instance, consider the example given in Figure B.1(a). If the denominator is distributed over the numerator, the result is the expression in Figure B.1(b). But if we factor the numerator first, the discovered factor of (x + 1) in the numerator cancels the (x + 1) in the denominator, leaving the simplified answer in Figure B.1(c). muMATH cannot take these decisions; it is a tool and not a problem solver. So certain variables are introduced into its environment. These variables are called control variables. Under the user's control, these variables are used to tell muMATH what manipulations to make.

Equation Solving

Here again, muMATH displays that it is not intelligent. It cannot solve equations of orders higher than three. It can solve, however, quadratic equations resulting in complex and/or real roots.

Trigonometric and Logarithmic Manipulations

It has the capabilities to manipulate logarithmic and trigonometric expressions. Manipulation of these expressions varies with the values of certain control variables. The functions available are:

LN (logarithm to the base e)

LOG (logarithm to other bases)

COS, TAN, COT, SEC, CSC, #E (for e), and #PI (for π)

Matrix Manipulations

The math system can also manipulate matrices. Matrices can be multiplied (or divided) by a matirx, or a scalar, transposed, inverted, and taken to integer power. It recognizes non-singular matrices (i.e. matrices without inverses). Attempts to invert a non-singular matrix would result in error messages equivalent to a division by zero error.

(a) $\dfrac{x^3 + x^2}{(x + 1)}$

(b) $\dfrac{x^3}{(x + 1)} \quad + \quad \dfrac{x^2}{(x + 1)}$

(c) $\dfrac{x^2(x + 1)}{(x + 1)} = x^2$

Figure B.1

The same firm that markets mu-MATH will provide an older system, PICOMATH, on a limited basis. PICOMATH is a symbolic mathematics program written in BASIC, to run primarily on microcomputers. Implementing this program on another system requires modifications to accommodate the basic facilities or features that may only be unique to the basic system in which it was written. This makes PICOMATH hard to transport, and so the origators have discontinued supporting it.

REDUCE

REDUCE is a LISP based computer-algebra system implemenented by Anthony Hearn and his colleagues for a variety of large computers. Currently, there are supported implementations for the PDP-10, PDP-20, IBM 360, IBM 370, Univac 1108, CDC Cyber, and Cray-1 machines, running under various popular operating systems. In its entirety, the system occupies about 400K bytes on an IBM 370, for which an additional minimum of at least 50K bytes is recommended as workspace. The system is modularly designed so that the users can save space by omitting unneeded packages. The system may also be accessed though ARPA Computer network for those who have access to the network accounts. Following is a brief summary of the built-in facilities:

* The system provides single-precision floating-point arithmetic as well as indefinite-precision rational arithmetic.

* Unavoidable algebraic transformations and optional ones controlled by flags are approximately similar to those of muMATH, except that REDUCE provices an important additional optional transformation. Cancellation of polynomial greatest divisors from the numerators and denominators fo rational expressions. REDUCE can perform such simplifications as the following:

$$\frac{2a^2x^2 - a^2b^2 - ax^3 + axb^2 - x^4 + bx^3}{a^2x^2 - a^2b - 2axb - ab^2 - bx^2 + b^2x} = \frac{2ax + ab + x^2}{a + b}$$

which might be overlooked by most people.

* There are some built-in exponential, logarithmic and trigonometric simplifications.

* Matrices having symbolic expressions as elements can be added, subtracted, multiplied, divided, and raised to integer powers, including inversion.

* There are facilities for solving the quantumelectrodynamics problems of the high-energy physics.

* There is a high-level surface programming language, which is essentially ALGOL, sweetened by modern control constructs such as WHILE loop, REPEAT loop, and CASE statement.

* Symbolic differentiation and integrations are built-in, and the latter is significantly more powerful than muMATH integrator, which merely uses a few elementary rules such as (17; pp.186):

$$\int (u + v)\, dx \quad \int udx + \int vdx,$$

$$\int cudx \rightarrow c\int udx \quad \text{if } c = \text{constant},$$

$$\int vf(u)dx \quad \frac{v}{du/dx} \quad f(u)du \quad \text{if } \frac{v}{du/dx} = \text{constant},$$

$$\int x^{-1} \rightarrow \ln x;$$

$$\int x^{x} \rightarrow \frac{x^{x+1}}{x} \quad \text{if } x = \text{const and } = -1,$$

$$\int \sin(x) \rightarrow \cos(x)$$

In contrast, extensive greatest-common-divisor, factorization, and linear-equation-solving support routines permit REDUCE to use the powerful new Risch-Norman integration algorithm. For a large class of integrands and solution basis functions, this algorithm is guaranteed to determine a closed-form solution if one exists, otherwise terminating with a guarantee that one does not exist.

* REDUCE provides a convenient pattern matcher, which provides a natural means for users to implement many extensions. To have the system automatically replace every instance of MC^2 by E, we can merely enter the rule:

```
LET M * C ** 2 = E;
```

Therefore, an expression such as 5 * M * C ** 3 + 8 would be replaced by 5 * E * C + 8. There is also a mechanism for letting pattern variables represent arbitrary subexpressions.

Most of REDUCE is written in a modular subset of itself called RLISP. In turn, RLISP is bootstrapped from standard LISP, which is a subset of many LISP implementations. RLSIP has the semanticcs of LISP clothed in the syntax of sweetened ALGOL. RLSIP is applicable not only to computer algebra, but also wherever LISP is applicable.

REDUCE was originally inspired by a desire to perform symbolic high-energy-physics computations which are far too arduous to do manually. Consequently, the internal representations of expressions reflect major concern with speed and storage efficiency for large expressions:

* In applied mathematics, the most numerous operations in very large expressions are usually addition, subtraction, multiplication, and exponentiation with positive integer exponents. There is frequently, at most, one division operation present, because

expressions are often put over a common denominator. If fractional powers, exponentials, logarithms, trigonometric functions or other irrational operations occur, they may usually be reduced to numerous repetitions of a few unnested distinct irrational functions having trivial arguments such as x, x+y, or $2\pi x$. Thus, polynomial operations account for most of the time and space. This suggests using a data structure oriented toward polynomials, thereby saving space and time by making the operator +, x, and $\uparrow$ implicit. This usual nature of large expressions also suggests storing irrational subexpressions uniquely, and treating them as additional variables with respect to any polynomial operations involving them.

* As the number of variables and their maximum degrees increase, a multivariate polynomial must have zero as a sharply increasing portion of its possible terms, in order to fit the polynomial into the computer memory. Moreover, the fit is possible only if the internal representation takes advantage of this sparsity. In general , we can avoid wasting space on itermediate-degree terms which are zero only if we explicitly store the exponents of the non-zero terms.

* Many multivariate polynomial algorithms are most concisely stated as univariate algorithms, recursively involving coefficients which are polynomials in at least one less variable.

* Classic multivariate polynomial division requires that one variable be distinguished as the leading variable and that the terms be accessible in decreasing order of degree.

In general, REDUCE is a program designed for general algebraic computations of interest to physicists, mathematicians and engineers. Its capabilities include:

1. expansion and ordering of polynomials and rational functions,
2. substitutions and pattern matching in a wide variety of forms,
3. automatic, and user controlled simplifications of rules,
4. calculations with symbolic matrices,
5. arbitary precision integer and real arithmetic,
6. facilities for defining new functions and extending program syntax,
7. analytic differentiation and integration,
8. factorization of polynomials,
9. Dirac matrix calculations of interest to high energy physicists.

The program is designed to run interactively if the operating system permits it or in batch mode otherwise.

MACSYMA

MACSYMA is a very large computer-algebra system implemented by the Mathlab group at the MIT Laboratory for Computer Science in Cambridge Massachussetts.

In its entirety, excluding the library of user-submitted routines, MACSYMA occupies 400,000 thirty-six bit words on the PDP-10. The system is modular, starting with a nucleus of 100,000 words. As perhaps implied by its name, MACSYMA provides more built-in math operations than any other computer-algebra system. Here are some highlights:

* The system provides arbitrary-precision floating-point as well as indefinite-precision arithmetic.

* Besides the usual unavoidable algebraic transformations, there are numerous optional automatic one controlled by flags or which are employed by applying specific functions to expressions. The most sophisticated of these transformations include cancellation of polynomial greatest common divisors, partial-fraction decomposition, nested polynomial decomposition such as completion of powers, and factorizations. For example, MACSYMA can perform the factorization (17; pp.189):

$$3w^2z^6 + 2w^3z^4 + 114xy^2z^3 - 10w^2y^2z^3 + 45w^2x^3z^3 - 3w^2z^3 + 76wxy^2z - 2w^3z -$$
$$380xy^4 + 1710x^4y^2 + 10w^2y^2 - 45w^2x^3 =$$
$$(3z^2+2wz - 10y^2 + 45x^3)(w^2y^3 + 38xy^2 - w^2).$$

* There are numerous built-in transformations for fractional powers, exponentials, logarithms, trigonometric functions, inverse trigonometric functions, hyperbolic functions and inverse hyperbolic functions. There are also transformations for some higher transcendental functions such as the error, gamma, beta, zeta, and psi functions.

* There are built-in matrix algebra on matrices having inspecified elements and unspecified sizes.

* There are special facilities for series analysis of periodic phenomena such as orbits.

* There is a high-level surface programming language which resembles ALGOL, with evidence of meta-LISP influence.

* Symbolic differentiation and integration are built-in. The latter employs a powerful RISCH algorithm, among other techniques. There is also a distinct program for definite integrals, which employs contour integration and other techniques besides indefinite integration.

* There is a powerful function which employs L'Hospital's rule and other techniques to compute limits.

* There are powerful functions for determining infinite and truncated generalized power-series expansions of expressions.

* There is a function which uses a variety of techniques to seek closed-form solutions to first-order and second-order ordinary differential equations.

* There is a built-in function which uses the powerful new Gosper algorithm to find closed forms for sums with indefinite or infinite summations limits. For example, the function is able to make the transformation:

$$\sum_{j=0}^{n} \frac{j^4 4^j}{\binom{2j}{j}} \longrightarrow \frac{2(n+1)(63n^4 + 112n^3 + 18n^2 - 22n + 3)4^n}{693 \binom{2n}{n}} - \frac{2}{231}$$

* Equations are legitimate expressions. Two equations or an equation and a non-equation can be added, multiplied, etc., and there is a powerful function named SOLVE which uses a variety of techniques to seek solutions to one or more simultaneous linear or non-linear equations. SOLVE is able to determine, as exact symbolic expressions involving "c", the four values of "x" which satisfy the quadratic equation:

$$x^4 = cx + 1$$

As another example, SOLVE is able to determine that the exact solutions for two simultaneous non-linear equations:

$$z^4 + x^2z^2 + xz^2 + y^2 + x^3 = 2yz^2 + x^2y + xy,$$
$$yz^2 + 2xyz + xy = 2xz^3 + 2x^2z + y^2$$

are the curve ($x = r$, $y = s^2$, $z = r$) together with the surface ($x = r$, $y = s^2 + r$, $z = s$), where "s" and "r" are arbitrary parameters.

* There is an extensive user-contributed program library which includes packages for vector and tensor analyses, ordinary and variational optimization, solution of integral equations, higher transcendental funcations, and dimemsional analysis.

Most of MACSYMA is written in MACLISP, which is a particulary elaborate version of LISP also developed at MIT. MACSYMA uses several internal representations, including Cambridge prefix and recursive polynomial representation somewhat like that of REDUCE. The major difference from the REDUCE polynomial representation is that in MACSYMA the variables are also implicit and stored separately only once per complete polynomial. This usually saves additional space in the expressions. Although the resulting algorithms are somewhat faster when combining polynomials having the same variables, there is some awkwardness or overhead involved in a preliminary padding phase when combining polynomials that do not have identical variables.

SCRATCHPAD

SCRATCHPAD is a very large computer-algebra system implementated at the IBM Thomas J. Watson Research Center. It is available there on an IBM 370, and it is also available from other IBM corporate sites via telephone. The system has not yet been released to the public.

In its entirety, the system occupies 1600K bytes on an IBM 370 with virtual storage for which an additional minimum of 100K bytes is recommended for workspace. The variety of built-in transformations currently lie between REDUCE and MACSYMA. However, each of the three systems has features that none of the others posses, and one of these features may be decisive advantage for a particular application. Here are some highlights of the SCRATCHPAD System:

* The system provides single-precision floating point arithmetic as well as indefinite-precision rational arithmetic.

* The built-in unavailable and optional algebraic transformation are approximately similarly to those of MACSYMA.

* The built-in exponential, logarithmic, and trigonometric transformation are approximately similarly to those of REDUCE.

* Besides built-in symbolic matrix-algebra, APL-like array operations are included, and they are even further generalized to permit symbolic operations of non-homogeneous arrays and on arrays of indefinite or infinite size.

* Symbolic differentiation and integration are built-in, with the latter employing the powerful Risch-Norman algorithm.

* There is a particularly elegant built-im facility for determining Taylor series expansions.

* There is a built-in Solve function capable of determining exact solution to a system of linear equations.

* There is a powerful pattern-matching facility which serves as the primary mechanism for user level extensions. The associated syntax is at a very high level, being the closest of all computer-algebra systems to the declarative, nonprocedural notation of mathematics. To implement the trigonometric multiple-angle expansions, we can merely enter the rewrite rules (17; pp. 191):

cos (n * x) = z * cos(x) * cos((n - 1) * x) -

cos ((n - z) *), n in (2, 3, . . .), x arb

sin (n * x) = 2 * cos (x) * sin ((n - 1) * x) -

sin ((n - 2) * x), n in (2, 3, . . .), x arb

Then, whenever we subsequently enter an expression such as cos (4 * b), the response will be a corresponding expanded expression such as:

$$8 \cos^4(b) - 8 \cos^2(b) + 1$$

Thus, programs resemble a collection of mathematical formulae, much as they would appear in a book or an article.

SCRATCHPAD has a particulary powerful yet easily used mechanism for controlling the output format expression. For example, the user can specify that an expression be displayed as power series in "x" with coefficients which are factored rational functions in "b" and "c", etc. For large expressions, such fine control over the output may mean the difference between an important new discovery and an incomprehensible mess.

APPENDIX C

Appendix C

General Instructions for Using the Program

The program SYMMATH will be moved to Dr. Allan's TSO faculty account. The source code listing put in a data binder will be placed in Dr. Allan's care. Since the listing will be several hundreds of pages when xeroxed, it was decided that it should not be bound with the thesis.

Once the interactive session has been initiated, the program will issue pertinent prompts for the user. It is quite self-explanatory. Therefore, this section is primarily concerned with telling the user how to get started.

TSO EXECUTION

1. Log on to Dr. Allan's TSO account.

2. Once in READY mode, type in the following:

   CALL SYMMATH.LOAD

   Then respond according to the prompts which the program issues forth.

BATCH EXECUTION

```
// △      Job Card
// △      EXEC PGM = SYMMATH
// STEPLIB DD DSN = T110030. SYMMATH.LOAD, DISP = SHR
// DATA.SYSIN DD *
          Input Data
/*
//SYSPRINT DD SYSOUT = A
//
```

For batch execution, please refer to Chapter II, Section B.3 for specific formats for the various problem types. Listed below are a selection of four sessions from four different problem types.

SELECTED SAMPLE SESSIONS

```
**********************************************
* THIS MODULE WILL ASSIST YOU IN SOLVING     *
* LINEAR EQUATIONS.  IT ATTEMPTS TO GIVE AS *
* MANY INTERMEDIATE STEPS AS POSSIBLE.       *
**********************************************


**********************************************
*  ENTER THE EQUATION YOU WISH TO SOLVE.    *
*  ENTER THE VARIABLE TO SOLVE FOR.         *
**********************************************


THE EQUATION ENTERED IS    2*(3*X-7)=3*(4*X+2)+10
NOW, SOLVE FOR    X


IDENTIFY THE LEFT HAND SIDE(LHS), AND THE
RIGHT HAND SIDE(RHS) OF THE EQUATION.

LHS = 2 ( 3 X - 7 )
-------------------

RHS = 3 ( 4 X + 2 ) + 10
------------------------


EXPAND THE LEFT HAND SIDE(LHS) TO GET:

LHS = 6 X - 14
--------------


EXPAND THE RIGHT HAND SIDE(RHS) TO GET:

RHS = 12 X + 16
---------------

SUBTRACT THE TERM NOT CONTAINING THE
VARIABLE TO BE SOLVED FOR ON THE LHS ( -14)
FROM BOTH SIDES TO GET:


LHS = 6 X
---------
RHS = 12 X + 30
---------------

SUBTRACT THE TERM CONTAINING THE
VARIABLE TO BE SOLVED FOR ON THE RHS
( X*12) FROM BOTH SIDES TO GET:


LHS =  - 6 X
------------
RHS = 30
```

IDENTIFY THE COEFFICIENT OF THE LHS
COEFFICIENT = -6

NOW, DIVIDE BOTH SIDES BY THE COEFFICIENT
TO GET THE FINAL SOLUTION:

X = - 5

****

* ENTER THE EQUATION YOU WISH TO SOLVE. *
* ENTER THE VARIABLE TO SOLVE FOR. *
****

THE EQUATION ENTERED IS 3*(2+X)=9-2*X
NOW, SOLVE FOR X

IDENTIFY THE LEFT HAND SIDE(LHS), AND THE
RIGHT HAND SIDE(RHS) OF THE EQUATION.

LHS = 3 (X + 2)

RHS = - 2 X + 9

EXPAND THE LEFT HAND SIDE(LHS) TO GET:

LHS = 3 X + 6

SUBTRACT THE TERM NOT CONTAINING THE
VARIABLE TO BE SOLVED FOR ON THE LHS (6)
FROM BOTH SIDES TO GET:

LHS = 3 X

RHS = - 2 X + 3

SUBTRACT THE TERM CONTAINING THE
VARIABLE TO BE SOLVED FOR ON THE RHS
(-X*2) FROM BOTH SIDES TO GET:

LHS = 5 X

RHS = 3

IDENTIFY THE COEFFICIENT OF THE LHS
COEFFICIENT = 5

```
          NOW, DIVIDE BOTH SIDES BY THE COEFFICIENT
          TO GET THE FINAL SOLUTION:

X = 3/5
-------

          *******************************************

          *  ENTER THE EQUATION YOU WISH TO SOLVE.   *
          *  ENTER THE VARIABLE TO SOLVE FOR.        *
          *******************************************

          THE EQUATION ENTERED IS   5*X+X+3=3*X-9
          NOW, SOLVE FOR   X


          IDENTIFY THE LEFT HAND SIDE(LHS), AND THE
          RIGHT HAND SIDE(RHS) OF THE EQUATION.

LHS = 6 X + 3
-------------

RHS = 3 X - 9
-------------

          SUBTRACT THE TERM NOT CONTAINING THE
          VARIABLE TO BE SOLVED FOR ON THE LHS ( 3)
          FROM BOTH SIDES TO GET:

LHS = 6 X
---------
RHS = 3 X - 12
--------------

          SUBTRACT THE TERM CONTAINING THE
          VARIABLE TO BE SOLVED FOR ON THE RHS
          ( X*3) FROM BOTH SIDES TO GET:

LHS = 3 X
---------
RHS =  - 12
-----------

          IDENTIFY THE COEFFICIENT OF THE LHS
          COEFFICIENT =  3

          NOW, DIVIDE BOTH SIDES BY THE COEFFICIENT
          TO GET THE FINAL SOLUTION:

X =  - 4
--------
          *******************************************

          *  ENTER THE EQUATION YOU WISH TO SOLVE.   *
          *  ENTER THE VARIABLE TO SOLVE FOR.        *
```

```
          ********************************************

          THE EQUATION ENTERED IS    X+5=0
          NOW, SOLVE FOR    X



          IDENTIFY THE LEFT HAND SIDE(LHS), AND THE
          RIGHT HAND SIDE(RHS) OF THE EQUATION.

LHS = X + 5
-----------

RHS = 0
-------

          SUBTRACT THE TERM NOT CONTAINING THE
          VARIABLE TO BE SOLVED FOR ON THE LHS ( 5)
          FROM BOTH SIDES TO GET:


LHS = X
-------
RHS =  - 5
----------

          IDENTIFY THE COEFFICIENT OF THE LHS
          COEFFICIENT =   1

          NOW, DIVIDE BOTH SIDES BY THE COEFFICIENT
          TO GET THE FINAL SOLUTION:

X =  - 5
--------
          ********************************************


          *  ENTER THE EQUATION YOU WISH TO SOLVE.     *
          *  ENTER THE VARIABLE TO SOLVE FOR.          *
          ********************************************


          THE EQUATION ENTERED IS    2*X+14=2
          NOW, SOLVE FOR    X



          IDENTIFY THE LEFT HAND SIDE(LHS), AND THE
          RIGHT HAND SIDE(RHS) OF THE EQUATION.

LHS = 2 X + 14
--------------

RHS = 2
-------

          SUBTRACT THE TERM NOT CONTAINING THE
          VARIABLE TO BE SOLVED FOR ON THE LHS ( 14)
          FROM BOTH SIDES TO GET:
```

LHS = 2 X

RHS = - 12

IDENTIFY THE COEFFICIENT OF THE LHS
COEFFICIENT = 2

NOW, DIVIDE BOTH SIDES BY THE COEFFICIENT
TO GET THE FINAL SOLUTION:

X = - 6

****

* ENTER THE EQUATION YOU WISH TO SOLVE. *
* ENTER THE VARIABLE TO SOLVE FOR. *
****

THE EQUATION ENTERED IS 4*X-9*X+22=3*(X+10)
NOW, SOLVE FOR X

IDENTIFY THE LEFT HAND SIDE(LHS), AND THE
RIGHT HAND SIDE(RHS) OF THE EQUATION.

LHS = - 5 X + 22

RHS = 3 (X + 10)

EXPAND THE RIGHT HAND SIDE(RHS) TO GET:

RHS = 3 X + 30

SUBTRACT THE TERM NOT CONTAINING THE
VARIABLE TO BE SOLVED FOR ON THE LHS (22)
FROM BOTH SIDES TO GET:

LHS = - 5 X

RHS = 3 X + 8

SUBTRACT THE TERM CONTAINING THE
VARIABLE TO BE SOLVED FOR ON THE RHS
(X*3) FROM BOTH SIDES TO GET:

LHS = - 8 X

RHS = 8

IDENTIFY THE COEFFICIENT OF THE LHS
COEFFICIENT = -8

NOW, DIVIDE BOTH SIDES BY THE COEFFICIENT
TO GET THE FINAL SOLUTION:

X = - 1

****

* ENTER THE EQUATION YOU WISH TO SOLVE. *
* ENTER THE VARIABLE TO SOLVE FOR. *
****

THE EQUATION ENTERED IS 4*(Y-9)=10*(2-Y)
NOW, SOLVE FOR Y

IDENTIFY THE LEFT HAND SIDE(LHS), AND THE
RIGHT HAND SIDE(RHS) OF THE EQUATION.

LHS = 4 (Y - 9)

RHS = - 10 (Y - 2)

EXPAND THE LEFT HAND SIDE(LHS) TO GET:

LHS = 4 Y - 36

EXPAND THE RIGHT HAND SIDE(RHS) TO GET:

RHS = - 10 Y + 20

SUBTRACT THE TERM NOT CONTAINING THE
VARIABLE TO BE SOLVED FOR ON THE LHS (-36)
FROM BOTH SIDES TO GET:

LHS = 4 Y

RHS = - 10 Y + 56

SUBTRACT THE TERM CONTAINING THE
VARIABLE TO BE SOLVED FOR ON THE RHS
(-Y*10) FROM BOTH SIDES TO GET:

LHS = 14 Y

```
----------
RHS = 56
--------

        IDENTIFY THE COEFFICIENT OF THE LHS
        COEFFICIENT =   14

        NOW, DIVIDE BOTH SIDES BY THE COEFFICIENT
        TO GET THE FINAL SOLUTION:

Y = 4
-----
        ********************************************

        *  ENTER THE EQUATION YOU WISH TO SOLVE.    *
        *  ENTER THE VARIABLE TO SOLVE FOR.         *
        ********************************************


        THE EQUATION ENTERED IS    3*M+5-M=2*(M-1)
        NOW, SOLVE FOR   M


        IDENTIFY THE LEFT HAND SIDE(LHS), AND THE
        RIGHT HAND SIDE(RHS) OF THE EQUATION.

LHS = 2 M + 5
-------------

RHS = 2 ( M - 1 )
-----------------


        EXPAND THE RIGHT HAND SIDE(RHS) TO GET:

RHS = 2 M - 2
-------------

        SUBTRACT THE TERM NOT CONTAINING THE
        VARIABLE TO BE SOLVED FOR ON THE LHS ( 5)
        FROM BOTH SIDES TO GET:


LHS = 2 M
---------
RHS = 2 M - 7
-------------

        SUBTRACT THE TERM CONTAINING THE
        VARIABLE TO BE SOLVED FOR ON THE RHS
        ( M*2) FROM BOTH SIDES TO GET:


LHS = 0
-------
RHS =  - 7
----------
     THIS IS THE FIRST OF THE TWO SPECIAL CASES
```

FOR FIRST DEGREE EQUATIONS. THE EQUATION IS A FALSE EQUATION

0= -7

```
*******************************************
*  ENTER THE EQUATION YOU WISH TO SOLVE.  *
*  ENTER THE VARIABLE TO SOLVE FOR.       *
*******************************************
```

THE EQUATION ENTERED IS 7- M=3*M+7-4*M
NOW, SOLVE FOR M

IDENTIFY THE LEFT HAND SIDE(LHS), AND THE RIGHT HAND SIDE(RHS) OF THE EQUATION.

LHS = - M + 7

RHS = - M + 7

SUBTRACT THE TERM NOT CONTAINING THE VARIABLE TO BE SOLVED FOR ON THE LHS (7) FROM BOTH SIDES TO GET:

LHS = - M

RHS = - M

SUBTRACT THE TERM CONTAINING THE VARIABLE TO BE SOLVED FOR ON THE RHS (M) FROM BOTH SIDES TO GET:

LHS = - 2 M

RHS = - 2 M

THIS IS THE SECOND OF THE TWO SPECIAL CASES FOR FIRST DEGREE EQUATIONS. THE EQUATION IS A TRUE EQUATION

-M*2= -M*2

```
*******************************************
*  ENTER THE EQUATION YOU WISH TO SOLVE.  *
*  ENTER THE VARIABLE TO SOLVE FOR.       *
*******************************************
```

```
***************************************************
* THIS MODULE WILL ASSIST IN DIFFERENTIATING      *
* FUNCTIONS.  IT DOES NOT DISPLAY THE             *
* INTERMEDIATE STEPS.                             *
***************************************************


**********************************************
* ENTER THE FUNCTION TO BE DIFFERENTIATED    *
* AND THE VARIABLE OF DIFFERENTIATION        *
* SEPERATED BY A COMMA.  FOR EXAMPLE:        *
* X**3 + X**2 + 10,X                         *
**********************************************


THE ORIGINAL FUNCTION IS:    X*SIN(X) + X**2


DIFFERENTIATE THIS FUNCTION WITH RESPECT TO    X


        2
EXPR = X   + X SIN ( X )
------------------------


DX = X COS ( X ) + 2 X + SIN ( X )
-----------------------------------


**********************************************
* ENTER THE FUNCTION TO BE DIFFERENTIATED    *
* AND THE VARIABLE OF DIFFERENTIATION        *
* SEPERATED BY A COMMA.  FOR EXAMPLE:        *
* X**3 + X**2 + 10,X                         *
**********************************************


THE ORIGINAL FUNCTION IS:    3*X**4 - 8*X**3 + 6*X**2 - 5*X + 18


DIFFERENTIATE THIS FUNCTION WITH RESPECT TO    X


          4        3        2
EXPR = 3 X   - 8 X   + 6 X   - 5 X + 18
----------------------------------------
```

```
          3          2
DRX = 12 X    - 24 X    + 12 X - 5
-------------------------------
```

```
*******************************************
* ENTER THE FUNCTION TO BE DIFFERENTIATED *
* AND THE VARIABLE OF DIFFERENTIATION     *
* SEPERATED BY A COMMA.  FOR EXAMPLE:     *
* X**3 + X**2 + 10,X                      *
*******************************************
```

```
THE ORIGINAL FUNCTION IS:    (2*X - 5)**3

DIFFERENTIATE THIS FUNCTION WITH RESPECT TO    X
```

```
                  3
EXPR = ( 2 X - 5 )
------------------
```

```
                   2
DRX = 6 ( 2 X - 5 )
-------------------
```

```
*******************************************
* ENTER THE FUNCTION TO BE DIFFERENTIATED *
* AND THE VARIABLE OF DIFFERENTIATION     *
* SEPERATED BY A COMMA.  FOR EXAMPLE:     *
* X**3 + X**2 + 10,X                      *
*******************************************
```

```
THE ORIGINAL FUNCTION IS:    3*X**8 - 4*X**6

DIFFERENTIATE THIS FUNCTION WITH RESPECT TO    X
```

```
          8       6
EXPR = 3 X   - 4 X
------------------
```

```
          7        5
DRX = 24 X   - 24 X
```

```
-------------------

       ********************************************
       * ENTER THE FUNCTION TO BE DIFFERENTIATED *
       * AND THE VARIABLE OF DIFFERENTIATION      *
       * SEPERATED BY A COMMA.  FOR EXAMPLE:      *
       * X**3 + X**2 + 10,X                       *
       ********************************************


       THE ORIGINAL FUNCTION IS:    (X**4 + 3)*(3*X**3 + 1)

       DIFFERENTIATE THIS FUNCTION WITH RESPECT TO    X


       4            3
XPR = ( X   + 3 ) ( 3 X   + 1 )
-------------------------------

      3          3          2       4
DX = 4 X   ( 3 X   + 1 ) + 9 X   ( X   + 3 )
--------------------------------------------


       ********************************************
       * ENTER THE FUNCTION TO BE DIFFERENTIATED *
       * AND THE VARIABLE OF DIFFERENTIATION      *
       * SEPERATED BY A COMMA.  FOR EXAMPLE:      *
       * X**3 + X**2 + 10,X                       *
       ********************************************


       THE ORIGINAL FUNCTION IS:   X/(X**2 - 3*X + 5)

       DIFFERENTIATE THIS FUNCTION WITH RESPECT TO    X


                2
XPR = X /  ( X   - 3 X + 5 )
-----------------------------

                             2              2           2
DX =  - X ( 2 X - 3 ) /  ( X   - 3 X + 5 )   + 1 / ( X   - 3 X + 5 )
--------------------------------------------------------------------
```

```
******************************************
* ENTER THE FUNCTION TO BE DIFFERENTIATED *
* AND THE VARIABLE OF DIFFERENTIATION     *
* SEPERATED BY A COMMA.  FOR EXAMPLE:     *
* X**3 + X**2 + 10,X                      *
******************************************
```

```
THE ORIGINAL FUNCTION IS:    (8*X)/(X**2 + 4)

DIFFERENTIATE THIS FUNCTION WITH RESPECT TO    X
```

```
                 2
EXP = 8 X /  ( X    + 4 )
-------------------------
```

```
             2          2        2           2
DX =  - 16 X    /  ( X    + 4 )    + 8 /  ( X    + 4 )
------------------------------------------------------
```

```
******************************************
* ENTER THE FUNCTION TO BE DIFFERENTIATED *
* AND THE VARIABLE OF DIFFERENTIATION     *
* SEPERATED BY A COMMA.  FOR EXAMPLE:     *
* X**3 + X**2 + 10,X                      *
******************************************
```

```
THE ORIGINAL FUNCTION IS:    (X**2 - 3*X + 5)**25

DIFFERENTIATE THIS FUNCTION WITH RESPECT TO    X
```

```
         2              25
EXPR = ( X   - 3 X + 5 )
-------------------------
```

```
            2              24
DX = 25 ( X   - 3 X + 5 )     ( 2 X - 3 )
------------------------------------------
```

```
*******************************************
* ENTER THE FUNCTION TO BE DIFFERENTIATED *
* AND THE VARIABLE OF DIFFERENTIATION     *
* SEPERATED BY A COMMA.  FOR EXAMPLE:     *
* X**3 + X**2 + 10,X                      *
*******************************************



THE ORIGINAL FUNCTION IS:   (X**4 - 5*X**3 + 3)**50

DIFFERENTIATE THIS FUNCTION WITH RESPECT TO   X


         4         3        50
EXPR = ( X   - 5 X   + 3 )
------------------------------


           4         3        49        3        2
DDX = 50 ( X   - 5 X   + 3 )      ( 4 X   - 15 X   )
----------------------------------------------------



*******************************************
* ENTER THE FUNCTION TO BE DIFFERENTIATED *
* AND THE VARIABLE OF DIFFERENTIATION     *
* SEPERATED BY A COMMA.  FOR EXAMPLE:     *
* X**3 + X**2 + 10,X                      *
*******************************************



THE ORIGINAL FUNCTION IS:   ((X**2 - 1)/(X**2 + 1))**5

DIFFERENTIATE THIS FUNCTION WITH RESPECT TO   X


          2       5       2       5
EXPR =  ( X   - 1 )   / ( X   + 1 )
-------------------------------------


                 2       5        2       6             2       4        2
DDX =  - 10 X ( X   - 1 )   /  ( X   + 1 )   + 10 X ( X   - 1 )   /  ( X   + 1 )
---------------------------------------------------------------------------------



*******************************************
* ENTER THE FUNCTION TO BE DIFFERENTIATED *
```

```
* AND THE VARIABLE OF DIFFERENTIATICN      *
* SEPERATED BY A COMMA.  FOR EXAMPLE:      *
* X**3 + X**2 + 10,X                       *
*******************************************
```

```
****************************************************
* THIS MODULE WILL ASSIST YOU IN UNDERSTANDING *
* HOW THE R-TH TERM OF A BINOMIAL EXPRESSION   *
* IS DETERMINED.                               *
****************************************************



ENTER THE BINOMIAL EXPRESSION, AND THE TERM
TO BE DETERMINED ACCORDING TO THIS FORMAT:
BINOMIAL EXPRESSION,TERM
FOR EXAMPLE: (X + Y)**3,2

THE BINOMIAL EXPRESSION ENTERED IS:    (X + Y)**8
THE TERM TO BE DETERMINED IS:    3


WRITE DOWN A FRACTION BAR AND TWO SETS
OF PARENTHESIS:


------------- ( )( )


PUT THE FIRST TERM OF THE BINOMIAL IN
THE FIRST SET OF PARENTHESIS, AND THE
SECOND TERM OF THE BINOMIAL IN THE SECOND
SET OF PARENTHESIS TO GET:


------------- (X )( Y)


THE COEFFICIENT WHICH REPLACES
THE FRACTION BAR IS THE COMBINATION
OF N CHOOSE R,  WHERE N IS THE EXPONENT
OF THE BINOMIAL, AND R IS THE TERM TO BE
DETERMINED.
THIS REDUCES TO:


FAC(N)             (X )  ( Y)
------------------
FAC(N-R)*FAC(R)


NOW PUT IN THE EXPONENTS


THE EXPONENT OF THE SECOND SET IS (R-1)
EXPONENT OF THE FIRST SET IS (N - (R-1))


UPONE EVALUATION OF THE ABOVE EXPRESSION
AFTER PUTTING IN THE EXPONENTS, WE GET:


                 3    5
RTHTERM = 56 X     Y
--------------------
```

```
ENTER THE BINOMIAL EXPRESSION, AND THE TERM
TO BE DETERMINED ACCORDING TO THIS FORMAT:
BINOMIAL EXPRESSION,TERM
FOR EXAMPLE: (X + Y)**3,2

THE BINOMIAL EXPRESSION ENTERED IS:    (X + Y)**8
THE TERM TO BE DETERMINED IS:    4


WRITE DOWN A FRACTION BAR AND TWO SETS
OF PARENTHESIS:


-------------( )( )


PUT THE FIRST TERM OF THE BINOMIAL IN
THE FIRST SET OF PARENTHESIS, AND THE
SECOND TERM OF THE BINOMIAL IN THE SECOND
SET OF PARENTHESIS TO GET:


-------------(X ) ( Y)


THE COEFFICIENT WHICH REPLACES
THE FRACTION BAR IS THE COMBINATION
OF N CHOOSE R,  WHERE N IS THE EXPONENT
OF THE BINOMIAL, AND R IS THE TERM TO BE
DETERMINED.
THIS REDUCES TO:


FAC(N)___________(X )  ( Y)
FAC(N- R)*FAC(R)


NOW PUT IN THE EXPONENTS


THE EXPONENT OF THE SECOND SET IS (R-1)
EXPONENT OF THE FIRST SET IS (N -  (R-1))


UPONE EVALUATION OF THE ABOVE EXPRESSION
AFTER PUTTING IN THE EXPONENTS, WE GET:


           4   4
R'TERM = 70 X    Y
------------------
```

```
ENTER THE BINOMIAL EXPRESSION, AND THE TERM
TO BE DETERMINED ACCORDING TO THIS FORMAT:
BINOMIAL EXPRESSION,TERM
FOR EXAMPLE: (X + Y)**3,2

THE BINOMIAL EXPRESSION ENTERED IS:    (X + Y)**8
THE TERM TO BE DETERMINED IS:   5


WRITE DOWN A FRACTION BAR AND TWO SETS
OF PARENTHESIS:


-------------( ) ( )


PUT THE FIRST TERM OF THE BINOMIAL IN
THE FIRST SET OF PARENTHESIS, AND THE
SECOND TERM OF THE BINOMIAL IN THE SECOND
SET OF PARENTHESIS TO GET:


-------------(X ) ( Y)


THE COEFFICIENT WHICH REPLACES
THE FRACTION BAR IS THE COMBINATION
OF N CHOOSE R,  WHERE N IS THE EXPONENT
OF THE BINOMIAL, AND R IS THE TERM TO BE
DETERMINED.
THIS REDUCES TO:


FAC(N)____________(X )  ( Y)
FAC(N-R)*FAC(R)


NOW PUT IN THE EXPONENTS


THE EXPONENT OF THE SECOND SET IS (R-1)
EXPONENT OF THE FIRST SET IS (N - (R-1))


UPONE EVALUATION OF THE ABOVE EXPRESSION
AFTER PUTTING IN THE EXPONENTS, WE GET:


           5   3
RHTERM = 56 X   Y
-------------------
```

```
ENTER THE BINOMIAL EXPRESSION, AND THE TERM
TO BE DETERMINED ACCORDING TO THIS FORMAT:
BINOMIAL EXPRESSION,TERM
FOR EXAMPLE: (X + Y)**3,2

THE BINOMIAL EXPRESSION ENTERED IS:    (X + 1)**8
THE TERM TO BE DETERMINED IS:    6


WRITE DOWN A FRACTION BAR AND TWO SETS
OF PARENTHESIS:


-------------- ( ) ( )


PUT THE FIRST TERM OF THE BINOMIAL IN
THE FIRST SET OF PARENTHESIS, AND THE
SECOND TERM OF THE BINOMIAL IN THE SECOND
SET OF PARENTHESIS TO GET:


-------------- (X ) ( 1)


THE COEFFICIENT WHICH REPLACES
THE FRACTION BAR IS THE COMBINATION
OF N CHOOSE R,  WHERE N IS THE EXPONENT
OF THE BINOMIAL, AND R IS THE TERM TO BE
DETERMINED.
THIS REDUCES TO:


FAC(N)_______________ (X )  ( 1)
FAC(N-R)*FAC(R)


NOW PUT IN THE EXPONENTS


THE EXPONENT OF THE SECOND SET IS (R-1)
EXPONENT OF THE FIRST SET IS (N - (R-1))


UPONE EVALUATION OF THE ABOVE EXPRESSION
AFTER PUTTING IN THE EXPONENTS, WE GET:


           6
R ITERM = 28 X
---------------
```

```
ENTER THE BINOMIAL EXPRESSION, AND THE TERM
TO BE DETERMINED ACCORDING TO THIS FORMAT:
BINOMIAL EXPRESSION,TERM
FOR EXAMPLE: (X + Y)**3,2

THE BINOMIAL EXPRESSION ENTERED IS:    (X + Y)**8
THE TERM TO BE DETERMINED IS:   6


WRITE DOWN A FRACTION BAR AND TWO SETS
OF PARENTHESIS:


-------------( )( )


PUT THE FIRST TERM OF THE BINOMIAL IN
THE FIRST SET OF PARENTHESIS, AND THE
SECOND TERM OF THE BINOMIAL IN THE SECOND
SET OF PARENTHESIS TO GET:


-------------(X )( Y)


THE COEFFICIENT WHICH REPLACES
THE FRACTION BAR IS THE COMBINATION
OF N CHOOSE R,  WHERE N IS THE EXPONENT
OF THE BINOMIAL, AND R IS THE TERM TO BE
DETERMINED.
THIS REDUCES TO:


FAC(N)____________(X )  ( Y)
FAC(N-R)*FAC(R)


NOW PUT IN THE EXPONENTS


THE EXPONENT OF THE SECOND SET IS (R-1)
EXPONENT OF THE FIRST SET IS (N -  (R-1))


UPONE EVALUATION OF THE ABOVE EXPRESSION
AFTER PUTTING IN THE EXPONENTS, WE GET:


            6    2
THTERM = 28 X    Y
-------------------
```

```
ENTER THE BINOMIAL EXPRESSION, AND THE TERM
TO BE DETERMINED ACCORDING TO THIS FORMAT:
BINOMIAL EXPRESSION,TERM
FOR EXAMPLE: (X + Y)**3,2
```

```
ENTER THE TWO POINTS, CONSECUTIVEY
IN THE FOLLOWING FORMAT: X1,Y1;X2,Y2



DETERMINE THE EQUATION OF A LINE DEFINED BY
     TWO POINTS:        (-2,-3) AND (-4,1);


THE EQUATION OF THE LINE IS:
          Y - Y1 = M(X - X1) IN POINT-SLOPE FORM


FIRST, WE MUST CALCULATE THE SLOPE
OF THE LINE USING THE TWO POINTS.



THE SLOPE OF A LINEAR FUNCTION IS GIVEN BY:

                                   DELTAY
                                   DELTAX


ONE MUST THEREFORE CALCULATE DELTAX, AND
DELTAY BEFORE CALCULATING THE SLOPE.


THE GIVEN POINTS ARE:
                      (-2,-3)   AND (-4,1)



CALCULATE DELTAX AS FOLLOWS:

          DELTAX = X2 - X1

                 = -4 -   -2


                 =   -2


CALCULATE DELTAY AS FOLLOWS:
          DELTAY = Y2 - Y1

                 = 1 - -3
                 =  4


NOW, CALCULATE THE SLOPE




        DELTAY
SLOPE = ------
        DELTAX
```

$$= \frac{4}{-2}$$

$$= -2$$

YOU MAY REWRITE THE SLOPE AS:

LOPE = - 2

NOW THAT WE HAVE CALCULATED THE SLOPE,
WE WILL PROCEED TO DETERMINE THE EQUATION
OF THE LINE.

$$Y - -3 = -2(X - -2)$$

EXPAND THE RIGHT HAND SIDE.

$$Y + 3 = -X*2-4$$

SUBTRACT THE RIGHT HAND SIDE FROM BOTH
SIDES TO GET THE FINAL EQUATION IN
STANDARD FORM.

$$Y + X*2 + 7 = 0$$

ENTER THE TWO POINTS, CONSECUTIVEY
IN THE FOLLOWING FORMAT: X1,Y1;X2,Y2

DETERMINE THE EQUATION OF A LINE DEFINED BY
TWO POINTS: (3,2) AND (5,6);

THE EQUATION OF THE LINE IS:
Y - Y1 = M(X - X1) IN POINT-SLOPE FORM

FIRST, WE MUST CALCULATE THE SLOPE
OF THE LINE USING THE TWO POINTS.

THE SLOPE OF A LINEAR FUNCTION IS GIVEN BY:

$$\frac{\text{DELTAY}}{\text{DELTAX}}$$

ONE MUST THEREFORE CALCULATE DELTAX, AND
DELTAY BEFORE CALCULATING THE SLOPE.

THE GIVEN POINTS ARE:
(3,2) AND (5,6)

CALCULATE DELTAX AS FOLLOWS:

DELTAX = X2 - X1

= 5 - 3

= 2

CALCULATE DELTAY AS FOLLOWS:
DELTAY = Y2 - Y1

= 6 - 2
= 4

NOW, CALCULATE THE SLOPE

$$\text{SLOPE} = \frac{\text{DELTAY}}{\text{DELTAX}}$$

$$= \frac{4}{2}$$

$$= 2$$

YOU MAY REWRITE THE SLOPE AS:

LOE = 2
-- -----

NOW THAT WE HAVE CALCULATED THE SLOPE,
WE WILL PROCEDE TO DETERMINE THE EQUATION
OF THE LINE.

Y - 2 = 2(X - 3)

EXPAND THE RIGHT HAND SIDE.

Y-2 = X*2-6

SUBTRACT THE RIGHT HAND SIDE FROM BOTH
SIDES TO GET THE FINAL EQUATION IN
STANDARD FORM.

Y-X*2 + 4 = 0

ENTER THE TWO POINTS, CONSECUTIVEY
IN THE FOLLOWING FORMAT: X1,Y1;X2,Y2

DETERMINE THE EQUATION OF A LINE DEFINED BY
TWO POINTS: (4,1) AND (6,4);

THE EQUATION OF THE LINE IS:
Y - Y1 = M(X - X1) IN POINT-SLOPE FORM

FIRST, WE MUST CALCULATE THE SLOPE
OF THE LINE USING THE TWO POINTS.

THE SLOPE OF A LINEAR FUNCTION IS GIVEN BY:

$$\frac{\text{DELTAY}}{\text{DELTAX}}$$

ONE MUST THEREFORE CALCULATE DELTAX, AND
DELTAY BEFORE CALCULATING THE SLOPE.

THE GIVEN POINTS ARE:
(4,1) AND (6,4)

CALCULATE DELTAX AS FOLLOWS:

DELTAX = X2 - X1

= 6 - 4

= 2

CALCULATE DELTAY AS FOLLOWS:
DELTAY = Y2 - Y1

= 4 - 1
= 3

NOW, CALCULATE THE SLOPE

$$\text{SLOPE} = \frac{\text{DELTAY}}{\text{DELTAX}}$$

$$= \frac{3}{2}$$

$$= (3/2)$$

YOU MAY REWRITE THE SLOPE AS:

LOE = 3/2
.. --------

NOW THAT WE HAVE CALCULATED THE SLOPE, WE WILL PROCEED TO DETERMINE THE EQUATION OF THE LINE.

Y - 1 = (3/2) (X - 4)

EXPAND THE RIGHT HAND SIDE.

Y- 1 = X*(3/2)-6

SUBTRACT THE RIGHT HAND SIDE FROM BOTH SIDES TO GET THE FINAL EQUATION IN STANDARD FORM.

Y- X*(3/2) + 5 = 0

ENTER THE TWO POINTS, CONSECUTIVEY
IN THE FOLLOWING FORMAT: X1,Y1;X2,Y2

DETERMINE THE EQUATION OF A LINE DEFINED BY
TWO POINTS: (1,3) AND (2,43);

THE EQUATION OF THE LINE IS:
Y - Y1 = M(X - X1) IN POINT-SLOPE FORM

FIRST, WE MUST CALCULATE THE SLOPE
OF THE LINE USING THE TWO POINTS.

THE SLOPE OF A LINEAR FUNCTION IS GIVEN BY:

$$\frac{\text{DELTAY}}{\text{DELTAX}}$$

ONE MUST THEREFORE CALCULATE DELTAX, AND
DELTAY BEFORE CALCULATING THE SLOPE.

THE GIVEN POINTS ARE:
(1,3) AND (2,43)

CALCULATE DELTAX AS FOLLOWS:

DELTAX = X2 - X1

= 2 - 1

= 1

CALCULATE DELTAY AS FOLLOWS:
DELTAY = Y2 - Y1

= 43 - 3
= 40

NOW, CALCULATE THE SLOPE


```
        DELTAY
SLOPE = ------
        DELTAX
```

$$= \frac{40}{1}$$

$$= 40$$

YOU MAY REWRITE THE SLOPE AS:

LOPE = 40

NOW THAT WE HAVE CALCULATED THE SLOPE, WE WILL PROCEDE TO DETERMINE THE EQUATION OF THE LINE.

Y - 3 = 40(X - 1)

EXPAND THE RIGHT HAND SIDE.

Y-3 = X*40-40

SUBTRACT THE RIGHT HAND SIDE FROM BOTH SIDES TO GET THE FINAL EQUATION IN STANDARD FORM.

Y-X*40 + 37 = 0

ENTER THE TWO POINTS, CONSECUTIVEY
IN THE FOLLOWING FORMAT: X1,Y1;X2,Y2

DETERMINE THE EQUATION OF A LINE DEFINED BY
TWO POINTS: (-2,1/2) AND (-1/2,3);

THE EQUATION OF THE LINE IS:
Y - Y1 = M(X - X1) IN POINT-SLOPE FORM

FIRST, WE MUST CALCULATE THE SLOPE
OF THE LINE USING THE TWO POINTS.

THE SLOPE OF A LINEAR FUNCTION IS GIVEN BY:

$$\frac{\text{DELTAY}}{\text{DELTAX}}$$

ONE MUST THEREFORE CALCULATE DELTAX, AND
DELTAY BEFORE CALCULATING THE SLOPE.

THE GIVEN POINTS ARE:
(-2,1/2) AND (-1/2,3)

CALCULATE DELTAX AS FOLLOWS:

DELTAX = X2 - X1

= -1/2 - -2

= (3/2)

CALCULATE DELTAY AS FOLLOWS:
DELTAY = Y2 - Y1

= 3 - 1/2
= (5/2)

NOW, CALCULATE THE SLOPE

$$\text{SLOPE} = \frac{\text{DELTAY}}{\text{DELTAX}}$$

$$= \frac{(5/2)}{(3/2)}$$

$$= (5/3)$$

YOU MAY REWRITE THE SLOPE AS:

SLOPE = 5/3

NOW THAT WE HAVE CALCULATED THE SLOPE,
WE WILL PROCEDE TO DETERMINE THE EQUATION
OF THE LINE.

$$Y - 1/2 = (5/3)(X - -2)$$

EXPAND THE RIGHT HAND SIDE.

$$Y - 1/2 = X*(5/3) + (10/3)$$

SUBTRACT THE RIGHT HAND SIDE FROM BOTH
SIDES TO GET THE FINAL EQUATION IN
STANDARD FORM.

$$Y - X*(5/3) - 23/6 = 0$$

```
     FOR FIRST DEGREE EQUATIONS.  THE EQUATION IS A FALSE EQUATION
      0= -7
          ********************************************


          *  ENTER THE EQUATION YOU WISH TO SOLVE.   *
          *  ENTER THE VARIABLE TO SOLVE FOR.        *
          ********************************************


          THE EQUATION ENTERED IS    7-M=3*M+7-4*M
          NOW, SOLVE FOR    M



          IDENTIFY THE LEFT HAND SIDE(LHS), AND THE
          RIGHT HAND SIDE(RHS) OF THE EQUATION.
LHS =  - M + 7
--------------

RHS =  - M + 7
--------------

          SUBTRACT THE TERM NOT CONTAINING THE
          VARIABLE TO BE SOLVED FOR ON THE LHS ( 7)
          FROM BOTH SIDES TO GET:


LHS =   - M
----------
RHS =   - M
----------

          SUBTRACT THE TERM CONTAINING THE
          VARIABLE TO BE SOLVED FOR ON THE RHS
          ( M) FROM BOTH SIDES TO GET:


LHS =  - 2 M
------------
RHS =  - 2 M
------------
     THIS IS THE SECOND OF THE TWO SPECIAL CASES FOR FIRST DEGREE
     EQUATIONS. THE EQUATION IS A TRUE EQUATION
      -M*2= -M*2
          ********************************************


          *  ENTER THE EQUATION YOU WISH TO SOLVE.   *
          *  ENTER THE VARIABLE TO SOLVE FOR.        *
          ********************************************
```

VITA

[redacted]

BIBLIOGRAPHY

1. Cain, R. D., A CAI Program for Aromatic Organic Synthesis Written in Time Sharing Fortran, Department of Chemistry, University of Kansas.

2. Chambers, J. A. College Teachers: Their Effect on Creativity of Students. J. Educational Psychology 65(1973), 326-334.

3. Chobot, R. B., and Holt, E., Case Study: Formative Evaluation of Computer Based Training Materials, JWK International Corporation.

4. Crawford, M. M. Design of a Computer System for Managing an Audio-Tutorial Laboratory, Master's Thesis, Virginia Commonwealth University.

5. FORMAC73 User's Manual, User Services, 128 Burruss Hall, Virginia Polytechnic Institute and State University.

6. Goodman, A. W., Analytical Geometry and the Calculus, Third Edition, MacMillan.

7. Howe, A., and Ramiszowski, A. J., Eds. International Yearbnook of Educational and Instructional Technology. 1978/1979, Nichols Pub., New York, 1978.

8. Kearsley G. P., The Cost of CAI: A matter of Assumption, AEDS J. 10, 3(1977), 100 - 110

9. Lata, A. J., An Interactive Time-Sharing Basic Tutorial Program Sequence in Introductory Electrochemistry, Department of Chemistry, University of Kansas.

10. Levien, R. E., The Emerging Technology, Instructional Uses of the Computer in Higher Education, McGraw-Hill, New York, 1972.

11. McKenzie, J. E. L. and Lewis, R., Interactive Computer Graphics in Teaching., Halsted Press, New York, 1978.

12. Newmyer, J., Gus, K., Intermediate Algebra, Second Edition, Merrill

13. Passman, B., The TICCIT Project; Sperry Univac Corporation, Blue Bell, Pennsylvania, June 1979.

14. Puvelle R., Rothstein, M., Fitch, J., Computer Algebra, Scientific American, December, 1981,
pp. 136-152

15. Pratt, T. W., Programming Languages: Design and Implementation, Prentice-Hall, 1975

16. Russo, P., Hanson, J. N., Some Data Conversion for Managing the internal and Output Form of Formac constants, Computer and Information Science Department, Cleveland State University.

17. Stoutemyer, D. R. Lisp Based Symbolic Mathematics Systems, Byte Magazine, August, 1979.

18. Sugarman, R. A. Second Chance Computer Aided Instruction. ICEE Spectrum (August, 1978), 29 - 37

19. Today's Problems, Tomorrow's Crises, A report of the National Science Board Commission in Pre-college Education in Mathematics, Science, and Technology.

20. Williams, G., The muSIMP/muMATH-79 Symbolic Mathematics System, Byte, November 1980.